Міністерство освіти і науки України

«Київський національний університет імені Тараса Шевченка»

Факультет інформаційних технологій

Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
завідувач кафедри кібербезпеки
та захисту інформації
_____ Н.В. Лукова-Чуйко
« » червня 2021р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

дипломної роботи
бакалавра
(назва освітнього рівня)

галузь знань 12 Інформаційні технології
(шифр і назва галузі знань)

спеціальність 125 Кібербезпека
(код і назва спеціальності)

освітня програма Кібербезпека
(назва освітньої програми)

на тему: «Ідентифікація загроз на основі пасивного аналізу трафіку мережі»

Виконавець: студент IV курсу, групи КБ-41

(підпис)	**Жолнер Іван Дмитрович** (прізвище ім'я по-батькові)

	Прізвище, ініціали	**Підпис**
Керівник	Мирутенко Л.В.	

Нормоконтроль	Даков С. Ю.	

Київ 2021

Міністерство освіти і науки України

«Київський національний університет імені Тараса Шевченка»

Факультет інформаційних технологій

Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
завідувач кафедри кібербезпеки
та захисту інформації
_____ Н.В. Лукова-Чуйко
«10» жовтня 2020 р.

ЗАВДАННЯ
на виконання дипломної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньої програми)

Студенту **КБ-41** (група) **Жолнеру Івану Дмитровичу** (прізвище ім'я по-батькові)

Тема дипломної роботи Ідентифікація загроз на основі пасивного аналізу трафіку мережі

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема дипломної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №2 від 08.10.2020 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Концепція пасивного мережевого аналізу, методи виявлення та ідентифікації загроз

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Необхідно ознайомитися з концепцією пасивного мережевого аналізу, методами виявлення та ідентифікації загроз; визначити недоліки при роботі мережевого аналізу; розробити механізм ідентифікації загроз, використовуючи класифікатори на основі нейронних мереж.

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність Розробка механізму ідентифікації загроз з використанням класифікаторів на основі нейронних мереж.

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання:12 жовтня 2020 року

Завдання видала		Л.В. Мирутенко
	(підпис)	(ініціали, прізвище)
Завдання прийняв до виконання		І. Д. Жолнер
	(підпис)	(ініціали, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Уточнення постановки задачі	25.01.2021 – 22.01.2020	*виконано*
2	Аналіз літератури	29.01.2020 – 11.02.2020	*виконано*
3	Обґрунтування вибору рішення	12.02.2020 – 15.02.2020	*виконано*
4	Аналіз особливостей використання пасивного мережевого аналізу	16.02.2020 – 04.03.2020	*виконано*
5	Аналіз систем виявлевлення та запобігання вторгнень	05.03.2020 – 21.03.2020	*виконано*
6	Дослідження методів класифікації мережевих загроз	22.03.2020 – 08.04.2020	*виконано*
7	Розробка механізму ідентифікації загроз	09.04.2020 – 10.05.2020	*виконано*
8	Оформлення пояснювальної записки	11.05.2020 – 27.05.2020	*виконано*
9	Підготовка до захисту дипломної роботи	28.05.2020 – 08.06.2021	*виконано*

Завдання видав		Л.В. Мирутенко
	(підпис)	(ініціали, прізвище)
Завдання прийняв до виконання		І. Д. Жолнер
	(підпис)	(ініціали, прізвище)

Термін подання дипломної роботи до ЕК 08 червня 2021 року

РЕФЕРАТ

Дипломна робота складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел, додатків, має 66 сторінок основного тексту, 2 таблиці та 2 формули. Список використаних джерел містить 40 найменування і займає 3 сторінки.

Метою даної роботи є розробка механізму ідентифікації вразливостей на основі пасивного аналізу трафіку мережі.

У роботі проаналізована сучасна науково-технічна література з данної тематики, виконано аналіз програмних рішень, розроблено механізм ідентифікації загроз з використанням класифікаторів на основі нейронних мереж.

Розроблений лістинг програми, що виконує ідентифікацію загроз по даним, що можуть бути отримані за допомогою пасивного мережевого аналізу, може використовуватися експертами з безпеки або для покращення систем виявлення вторгнень.

Ключові слова: ідентифікація загроз, пасивний мережевий аналіз, класифікатори, нейронні мережі.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

IDS	–	Intrusion Detection System
IANA	–	Internet Assigned Numbers Authority
RFC	–	Request for Comments
TCP / IP	–	Transport Control Protocol / Internet Protocol
SYN	–	Synchronize sequence numbers
HTTP	–	HyperText Transfer Protocol
ACL	–	Access Control List
SPI	–	Shallow Packet Inspection
MPI	–	Middle Packet Inspection
DPP	–	Deep Packet Processing
IPS	–	Intrusion Prevention System
NIDS	–	Network Intrusion Detection System

ЗМІСТ

ВСТУП

Актуальність даної роботи визначається тим, що на сьогоднішній день практично кожен користувач комп'ютера використовує мережі загального користування, тому може бути підвержений діям зловмисників. Пасивний аналіз мережі дозволяє викривати такі дії та дізнаватись причини їх виникнення.

Розвиток мережевих технологій призводить до ускладнення структури мереж та збільшення їх розмірів, а також до потреби розробки інструментів, що дозволять як локалізувати проблеми, які виникають у мережах, так і аналізувати причини їх появи. Таку задачу на сьогоднішній день вирішують мережеві аналізатори, перевагами використання яких є можливість проводити накопичення, обробку, класифікацію, контроль і модифікацію мережевих пакетів в залежності від їх вмісту в реальному часі.

Тому *метою роботи* є розробка механізму ідентифікації загроз на основі пасивного аналізу трафіку мережі.

Для досягнення поставленої мети необхідно вирішити такі *завдання*:

1. Проаналізувати методи пасивного мережевого аналізу, його механізми роботи, його використання в системах безпеки.
2. Визначити методи класифікації мережевих загроз використовуючи перехоплений трафік.
3. Порівняти ефективність методів класифікації мережевих загроз та визначити найкращій метод, який можна використовувати для виконання поставленого завдання.
4. Підвищити ефективність методів класифікації шляхом модифікації данних, що використовуються обраними методами.
5. Розробити механізм визначення мережевих загроз, використовуючи методи класифікації заснованих на нейронних мережах.

Об'єктом дослідження є процес ідентифікації загроз на основі пасивного аналізу мережі.

Предметом дослідження є методи класифікації мережевих загроз.

Методи дослідження, що використовувались при написанні дипломної роботи:

- абстрагування;
- аналіз;
- порівняння;

Практична цінність є використання розробленого механізму ідентифікації загроз за допомогою класифікаторів на основі нейронних мереж для удосконалення IPS.

РОЗДІЛ 1
ОСОБЛИВОСТІ ВИКОРИСТАННЯ ПАСИВНОГО МЕРЕЖЕВОГО АНАЛІЗУ

Пасивний аналіз мережевого трафіку – це сукупність технологій, що дозволяють проводити накопичення, обробку, класифікацію, контроль і модифікацію мережевих пакетів в залежності від їх вмісту в реальному часі [1].

Пасивний метод аналізу мережевого трафіку схожий на активний. Відмінність у тому, що активний формує запроси у мережеві вузли, а пасивний просто збирає відповіді, які виходять від активних вузлів мережі. У порівнянні з активним методом аналізу мережі, пасивний має ряд переваг.

Переваги пасивного методу аналізу перед активним:

- відсутність генерування трафіку;
- не провокує системи виявлення вторгнень (IDS);
- можливість характеризації хостів;

Першим є відсутність генерування трафіку в мережі при аналізі, що позитивно позначається на пропускній спроможності, особливо, якщо вона невелика. Також пасивний аналіз не провокує системи виявлення вторгнень, так як працює, аналізуючи природний, вже згенерований системою, трафік. Крім того, він здатний в деяких випадках виявляти брандмауери і характеризувати хости. Однак пасивний метод аналізу трафіку також має і деякі недоліки.

Недоліки пасивного методу аналізу перед активним:

- незручне розгортання;
- потреба у вдалому розташуванні сенсорів;
- великий час обробки подій спеціалістами по безпеці;

Головний недолік пасивного методу - це порівняно незручне розгортання системи аналізу, так як від адміністратора потребується розташувати сенсори так, щоб через нього проходив корисний трафік.

Виділяється корисність техніки пасивного аналізу при запобіганні атак хакерів або шкідливого програмного забезпечення. Аналіз трафіку адміністратором, може допомогти, згодом, знайти джерела паразитного трафіку, причиною якого може бути і атака. Також позитивну роль пасивний аналіз зіграє і при реагуванні на злом або атаку. У такій ситуації пасивний аналіз дозволить дати оцінку ситуації і отримати можливу відповідь на питання про реалізацію і проведенні атаки.

Пасивний аналіз може стати ключовою ланкою в організації захисту від витоку інформації. Це можливо завдяки постійному контролю пакетів, при цьому контролю з боку людини, що істотно знижує помилки і помилкові спрацьовування.

Пасивні методи аналізу можуть також багато чого сказати про мережу і про те, як вона працює. Без чіткого розуміння інфраструктури дуже складно розробити ефективну політику в області безпеки. Знаючи адресний простір, коли користувач отримує доступ в Інтернет з мережі або з операційної системи, встановленій в мережі, можна оцінити можливість впливу уразливості і наслідки з точки зору безпеки мережі. Або, використовуючи мережеві аналізатори правильно розмістити у брандмауери і IDS-датчики у мережі.

Пасивний аналіз дозволяє виявити невирішені служби та інші аномальні поведінки користувачів в мережі практично миттєво. Простий збір пакетів за допомогою Wireshark або будь-якого іншої програми спостереження за мережею буде визначати наявність передачі потоків даних, відкритих файлів в Peer-to-peer мережах, ігрової активності та інших джерел несанкціонованого використання мережі. Можливе використання Wireshark з фільтром пакетів на певні IP-адреса внутрішньої мережі, а потім відсортувати по портах TCP або UDP. У більшості випадків, побачимо загальний набір служб, які легко ідентифікувати. Такі TCP порти, як правило, використовуються операційними системами для роботи в мережі і типовими службами (DNS, FTP, HTTP і тд.).

Пасивні аналізатори є, також, гарним інструментом при реагуванні на інциденти. Зловмисники не освідомлені у архітектурі та побудові мережі, вони тільки її використовують. У більшості випадків виконуваний код зловмисника працює за тим же принципом. Моніторинг мережі в режимі реального часу дозволяє

визначати зону копроментації, які системи можуть бути скомпрометовані нападом і, як цей напад стався [2].

1.1 Аналізатори трафіку.

Для отримання більш чіткого уявлення про аналізатори трафіку, роздивимося аналізатор трафіку Wireshark, який спрощує аналіз пакетів адміністратором мережі. Програма швидко класифікує пакети і розподіляє по групах. Реалізовані фільтри, що дозволяють зробити вибірку пакетів, спираючись на певні характеристики. Трафік аналізується безпосередньо системним адміністратором або спеціалістом по безпеці, і программа підтримує перегляд того які служби працюють з якими портами. Кожний додаток зазвичай працює зі стандартним для неї портом, такі порти призначаються IANA (Internet Assigned Numbers Authority) і описані в RFC для TCP / IP служб.

Wireshark - програма-аналізатор трафіку для комп'ютерних мереж. Має графічний користувальницький інтерфейс.

Функціональність, яку надає Wireshark, дуже схожа з можливостями програми tcpdump, однак Wireshark має графічний користувальницький інтерфейс і набагато більше можливостей із сортування та фільтрації інформації. Програма дозволяє користувачеві переглядати весь проходить по мережі трафік в режимі реального часу, переводячи мережеву карту в нерозбірливий режим (Promiscuous mode).

Wireshark має багатий набір функцій, який включає в себе наступне:

- глибока перевірка сотень протоколів;
- захоплення в режимі реального часу та аналіз у режимі офлайн;
- стандартний трипанельний браузер пакетів;
- працює на Windows, Linux, macOS, Solaris, FreeBSD, NetBSD та ін.;
- перехоплені мережеві дані можна переглядати за допомогою графічного інтерфейсу користувача або за допомогою утиліти TShark в режимі TTY;
- потужні фільтри відображення;
- розширений VoIP-аналіз;

- читання / запис багатьох різних форматів файлів захоплення;
- файли захоплення, стиснуті за допомогою gzip;
- дані в реальному часі можна читати з Ethernet, IEEE 802.11, PPP / HDLC, ATM, Bluetooth, USB, Token Ring, Frame Relay, FDDI та ін.;
- підтримка розшифровки багатьох протоколів, включаючи IPsec, ISAKMP, Kerberos, SNMPv3, SSL / TLS, WEP та WPA / WPA2 [3];

Також за допомогою мережевих аналізаторів існує можливість отримати і деяку інформацію про операційну систему яка інстальована на хості. Це можливо завдяки тому, що TCP / IP fingerprinting працює практично ідентично як в пасивному, так і в активному режимах. Можливість визначити ОС дає той факт, що в різних системах по-різному реалізований TCP / IP стек.

Крім того можна виділити аналізатор трафіку під назвою P0f fingerprinting, який використовує пасивний аналіз мережі для ідентифікації користувачів.

P0f - це інструмент, який використовує безліч складних, суто пасивних механізмів fingerprinting, щоб ідентифікувати користувачів, які стоять за будь-якими випадковими зв'язками TCP / IP, не втручаючись ніяким чином. Він включає значну кількість удосконалень fingerprinting на рівні мережі та вводить можливість міркувати щодо корисних навантажень на рівні додатків (наприклад, HTTP).

Є чотири параметри, які постійно змінюються в залежності від встановленої операційної системи, до них відносяться TOS, TTL, DF. Саме аналіз цих значень дозволяє нам судити про працюючу на хості ОС. Програма p0f 2.0 ці параметри розширює і дає ще більш точний і чіткий результат

Деякі можливості p0f включають:

- високо масштабована та надзвичайно швидка ідентифікація операційної системи та програмного забезпечення на обох кінцевих точках ванільного TCP-з’єднання – особливо в налаштуваннях, де зонди Nmap заблоковані, занадто повільні, ненадійні або просто спрацьовують сигнали тривоги IDS;
- автоматичне визначення спільного використання з'єднань / NAT, балансування навантаження та налаштування проксі-сервера на рівні програми;

• виявлення клієнтів та серверів, які підробляють декларативні заяви, такі як X-Mailer або User-Agent;

• інструмент може працювати на передньому плані або як демон, і пропонує простий API у режимі реального часу для сторонніх компонентів, які хочуть отримати додаткову інформацію про акторів, з якими вони спілкуються.

Загальне використання p0f включає:

• розвідку під час випробувань на проникнення;

• рутинний моніторинг мережі;

• виявлення несанкціонованих мережевих з'єднань у корпоративному середовищі;

• подання сигналів для засобів запобігання зловживанням [4];

1.2 Аналіз мережевих пакетів

Для ідентифікації потенційно небезпечного трафіку необхідне використання інструменів для перегляду вмісту пакетів, для подальшого прийняття рішення стосовно шкідливості переглянутого пакету.

Виконавши аналіз літератури [1, 2] можна зробити висновок, що розвиток технологій аналізу мережевого трафіку розвивався у напрямку «глибини», тобто, підвищення для окремого мережевого пакета рівня моделі OSI, дані якого аналізуються.

1.2.1 Поверховий аналіз пакетів (SPI)

Поверхневий аналіз пакетів (SPI) відноситься до перевірки заголовків пакетів для оптимізації маршрутизації пакетів, виявлення зловживання мережею та статистичного аналізу. Така перевірка не розкриває вміст пакетів даних.

Це менш досконала версія техніки глибокої перевірки пакетів (DPI), яка може бути використана для блокування пакетів на основі їх вмісту. На відміну від DPI, SPI робить загальні загальні відомості про трафік, засновані виключно на оцінці

заголовка пакета. Незважаючи на те, що неглибока перевірка пакетів не може надати таких детальних оцінок трафіку, як DPI, вона набагато краще справляється з більшим обсягом пакетів, ніж DPI.

SPI є менш досконалий, ніж DPI, але він здатний обробляти більший обсяг трафіку набагато швидше. SPI схоже на судження про книгу за її обкладинкою. Цей метод є більш популярний серед користувачів, оскільки DPI дозволяє маскувати пакети під інший вид трафіку.

Це технологія аналізу трафіку, яка грунтується виключно на заголовках пакету рівнів 1-3 (фізичний, канальний, мережевий) по моделі OSI. Має низькі вимоги до обчислювальних ресурсів, що дозволяє аналізувати великі обсяги трафіку. Технологія широко поширена, на її основі працює більшість міжмережевих екранів операційних систем (зокрема в ОС Windows XP / Vista і OS X), маршрутизаторів і інших мережевих пристроїв. На її основі реалізовані мережеві списки контролю доступу на рівні IP адрес і портів (ACL). Таким чином, дана технологія добре підходить для розмежування доступу ззовні до окремих комп'ютерів і сервісів внутрішньої мережі.

SPI раніше дбає про конфіденційність вхідного пакету даних, він ніколи не розкриває вміст пакетів даних [5]. Неглибока перевірка пакетів - це легкий метод перевірки пакетів серед усіх трьох (SPI, MPI, DPI). Неглибока перевірка пакетів працює за умови, що мережеві атаки поводяться по суті інакше, ніж звичайні потоки даних [6].

Одним з перших методів перевірки потоків даних був заснований на інформації, що міститься в заголовках пакетів, більш конкретно на IP-адресах, а також портах відповідних мережевих додатків [7]. Інспекція дрібної перевірки пакетів, в основному, використовується для виявлення вихідної IP-адреси, IP-адреси призначення, номера порту джерела, номера порту призначення та імені протоколу вхідного пакету даних. Маючи номер порту пакета даних, SPI використовує для ідентифікації імена програми. Виявивши ім’я програми з трафіку у реальному часі, SPI використовує відповідність імені програми з присвоєним Інтернетом номером (IANA) іменем програми цього конкретного порта. Якщо обидві назви програми

збіглися, тоді SPI вирішив, що вхідний трафік є безпечним і захищеним в іншому ж випадку, SPI вирішує, що пакет даних є зловмисним. Раніше номери портів були незмінними або статичними, але в наш час номери портів можна змінювати і використовуються динамічні номери портів, з цієї причини в даний час SPI не корисна, оскільки весь час вона не показує точного результату. Більш того, функція дрібної інспекції пакетів може виконувати лише роботу з інспекції пакетів у моделі передачі даних та фізичному рівні моделі взаємодії відкритої системи (OSI). Отже, інспекція дрібних пакетів не є корисною технікою перевірки трафіку в даний час.

1.2.2 Середній аналіз пакетів (MPI)

Це технологія аналізу трафіку, яка грунтується на інспектуванні сесій і сеансів зв'язку та ініційованих додатком, але з посередніком. В таких випадках також застосовується термін «проксі додатків» (application proxy). В рамках даної технології вміст пакетів аналізується частково і по визначеним правилам. Не використовуються складні методи аналізу такі як сигнатурний.

На основі посередницьких вузлів, що називаються середніми, які розміщуються по всій мережі, здійснюється перевірка пакету за допомогою спеціальних програм, що працюють на цих пристроях, здатних перехоплювати інформацію заголовка. За допомогою проміжних пристроїв або вузлів у мережі середня інспекція пакетів використовувалась для моніторингу та аналізу вхідних та вихідних пакетів даних із цих середніх блоків або вузлів. Інспекція середніх пакетів працює як шлюз між комп’ютерами кінцевого користувача та Інтернет-провайдером. Середня інспекція пакетів може бути використана для державного нагляду. Середня інспекція пакетів може виконувати лише роботу з перевірки пакетів у транспортній, мережевій, лінійній передачі даних та фізичному рівні моделі взаємозв’язку відкритих систем (OSI). Використовуючи перевірку середніх пакетів, адміністратори мережі в основному обмежують користувачів мережі завантажувати чи отримувати шкідливий контент через Інтернет.

Пристрої, що реалізують даний функціонал розміщуються між провайдером інтернету і кінцевим користувачем. Дані пристрою розбирають заголовки аж до транспортного рівня і невелику частину даних пакета для зіставлення розібраної частини з деяким списком розбору (parse list). Дані списки зазвичай коротше списків ACL і надають більш широкий діапазон дій на відміну від «allow/forbid» у випадку з ACL. Ці списки також більш виразні, так як дозволяють прив'язуватися не до IP-адрес, а до формату даних пакетів і даних деяких протоколів рівня додатків, наприклад, URL-адресами в разі протоколу HTTP. За допомогою MPI можна, заблокувати можливість отримання flash-файлів або картинок з певних інтернет сервісів (на рівні уявлення OSI) або заблокувати частину команд (на рівні додатку OSI) в окремих протоколах. Набір протоколів, дуже обмежений. В перших версіях CheckPoint FireWall-1 підтримувалися протоколи Telnet, FTP, HTTP, а в Cisco Private Internet Exchange - FTP, HTTP, H.323, RSH, SMTP і SQLNET. Згодом дані набори незначно розширювалися. Також відомо, що дана технологія використовується в продуктах компаній McAfee і Symantec. Міжмережеві екрани, що використовують цю технологію, відносяться до другого покоління [8].

1.2.3 Глибокий аналіз пакетів (DPI)

Високошвидкісний та постійний доступ до мережі є звичним явищем у всьому світі, що створює попит на більш досконалу обробку пакетів та підвищену безпеку мережі. Відповідь на цю складну мережеву обробку та мережеву безпеку може надати Deep Packet Inspection (DPI) [9]. По суті, глибока перевірка пакетів дозволяє точно класифікувати та контролювати трафік з точки зору вмісту та програм. Іншими словами, він аналізує вміст пакетів та забезпечує обробку вмісту. Найбільш складним завданням у DPI є перевірка вмісту, оскільки тіло (корисне навантаження) кожного пакета повинно бути відсканованим [10, 11]. Загалом системи DPI повинні забезпечувати наступне:

- висока пропускна здатність обробки;
- низька вартість впровадження;

- гнучкість у модифікації та оновлення описання вмісту;
- масштабованість із збільшенням кількості описання вмісту;

Вищезазначених цілей стає важче досягти через дві причини. По-перше, розрив між пропускною здатністю мережі та обчислювальною потужністю зростає [12]. По-друге, база даних відомих моделей атак стає більшою та складнішою. В даний час кілька мережевих функцій потребують більш ефективного аналізу та інформації про вміст та дані додатків обробних пакетів.

DPI використовується в мережевих додатках, таких як:

• *Системи виявлення / запобігання вторгненню в мережу (IDS / IPS):* на відміну від традиційних брандмауерів, NIDS / NIPS сканують весь корисний набір пакетів на наявність шаблонів, що вказують на небезпечний вміст. Комбінація класифікації пакетів (узгодження заголовків) та перевірки вмісту використовується для ідентифікації відомих описів атак. Попередні методи, такі як перевірка стану, все ще необхідні для забезпечення ефективної безпеки.

• *Перемикачі 7го рівня OSI:* автентифікація, балансування навантаження, фільтрація на основі вмісту та моніторинг - це функції, які підтримують комутатори 7го рівня. Наприклад, веб-комутатори, обізнані з програмами, забезпечують прозоре та масштабоване балансування навантаження в центрах обробки даних.

• *Керування трафіком та маршрутизація:* маршрутизація та керування трафіком на основі вмісту можуть диференціювати класи трафіку на основі даних програми.

DPP (Deep Packet Processing) – це технологія аналізу пакетів що може робити такі дії над пакетами, як модифікація, фільтрація або перенаправлення [6]. Дана технологія є логічним розвитком MPI. В рамках даного підходу аналізатор переглядає вміст кожного пакета повністю. Одним з важливих відмінностей від попередніх технологій є те, що системи на базі DPI може приймати рішення не тільки по вмісту пакетів, але і за непрямими ознаками, властивим якимось певним мережевим програмами і протоколам.

Для цього може використовуватися статистичний аналіз. Наприклад, аналіз частоти зустрічі певних символів, довжин пакетів, відстань між мітками часу

оследовательних пакетів і т.д. Також, в порівнянні з попередніми підходами, значно розширено список застосувань технології: класифікація, обмеження смуги, пріоритезація, кешування і тд. Технологія DPI отримала розвиток, через стрімке зростання обчислювальних спроможностей процесорів, їх швидкодії і можливостей для більш повного і точного аналізу мережевих даних. На відміну від MPI, дана технологія спочатку розроблялася для високошвидкісної обробки та ідентифікації великої кількості додатків в реальному часі.

Таким чином, рішення на основі DPI добре масштабується як по ширині мережевого каналу, так і за кількістю ідентифікованих додатків. З точки зору реалізації, основний компонент будь-якого рішення DPI - модуль класифікації, що відповідає за класифікацію мережевих потоків.

При цьому в залежності від цілей застосування DPI, класифікація може виконуватися з різною точністю:

- тип протоколу або додатка (наприклад, Web, P2P, VoIP)
- конкретний протокол на рівні додатка (HTTP, BitTorrent, SIP)
- додаток, що використовує власний протокол (Google Chrome, µTorrent, Skype)

Технологія DPI на даний момент є поточним стандартом для засобів аналізу мережевого трафіку і відноситься до області критично важливих технологій необхідних для забезпечення мережевої безпеки.

Внаслідок цього останнім часом на міжнародному рівні було прийнято низку стандартів, вимог і рекомендацій щодо особливостей реалізації, внутрішньою будовою та набору функцій відповідних коштів.

Технології глибокої інспекції пакетів (DPI) призначені для того, щоб оператори мережі могли точно визначити походження та вміст кожного пакету даних, що проходить через мережеві концентратори [13]. Глибока інспекція пакетів, що використовується переважно постачальниками послуг Інтернету для аналізу високошвидкісного мережевого трафіку, для державного нагляду, а також для захисту кінцевого користувача від зловмисної діяльності. Пакети даних перевіряються на наявність конкретних підписів протоколів для ідентифікації

мережевих програм. DPI здатний виявляти протоколи та програми за допомогою трьох методів, а саме виявлення портів, виявлення підписів та евристичним методом. У більшості програм DPI використовує підхід виявлення підпису для узгодження підписів за допомогою автоматичного узгодження шаблонів.

DPI використовує два підходи для збору пакетів даних: дзеркальне відображення портів та оптичний спліттер. Інспекція глибоких пакетів може виконувати завдання моніторингу всіх рівнів моделі взаємозв'язку відкритих систем (OSI), що є величезною перевагою глибокої інспекції пакетів над дрібною інспекцією пакетів та інспекцією середніх пакетів. DPI може перевіряти заголовок пакета даних, а також корисне навантаження пакета даних, і ця характеристика долає недоліки дрібної перевірки пакетів та перевірки середніх пакетів, з цієї причини ступінь точності DPI виявлення шкідливих дій вищий, ніж SPI та MPI, оскільки DPI раніше виконує роботу з інспекції вмісту пакету.

1.3 Функціональні особливості систем виявлення / запобігання вторгнень (Intrusion Detection / Prevention Systems)

Для ідентифікації та запобігання вразливостей в мережі потрібно вчасно визначати спроби неправомірних або шкідливих дій. Також необхідно виконувати постійний аналіз внутрішнього мережевого трафіку та вдосконалювати структуру або вразливості учасників мережі.

Такі цілі можливо виконувати за допомогою систем виявлення / запобігання вторгнень, а в нашому випадку саме NIDS (Network Intrusion Detection System). Вони розташовуються в стратегічному місці або у таких місцях мережі, де можливо контролювати трафік усіх пристроїв у мережі. Здійснюється контроль усього трафіку даних всієї підмережі та порівнення трафіку, який передається у підмережі з бібліотекою відомих атак. При цьому можливе удосконалення роботи NIDS за допомогою нейронних мереж, що будуть коасифікувати трафік та виділяти в ньому потенційно небезпечні. Як тільки атака буде визначена як потенційно небезпечна

або визначено відхилення у поведінці, буде відсилатися попередження адміністратору.

NIDS-система може контролювати велике число TPC-запитів на з'єднання (SYN) з багатьма портами на обраному комп'ютері, виявляючи, таким чином, що хтось намагається здійснити сканування TCP — портів. Мережева IDS може запускатися або на окремому комп'ютері, який контролює свій власний трафік, або на виділеному комп'ютері, прозоро переглядають весь трафік у мережі (концентратор, маршрутизатор) [14].

На сьогодняшній день брандмауери широко використовуються для запобігання доступу до систем з усіх точок доступу (портів), але вони не можуть усунути всі загрози безпеці, а також не можуть виявити атаки коли вони трапляються. Брандмауери здатні зрозуміти деталі протоколу, які перевіряються, відстежуючи стан з'єднання. Вони фактично встановлюють та контролюють зв’язки, поки вони не припиняються. Однак сучасні потреби в безпеці мережі вимагають набагато ефективнішого аналізу та розуміння даних програми [15]. Загрози та проблеми безпеки на основі вмісту виникають частіше, щодня. Передавання вірусів, спам, підробка електронної пошти та небезпечні або небажані дані все більше дратують і викликають проблеми. Тому брандмауери повинні підтримувати властивості глибокої перевірки пакетів, щоб забезпечити захист від цих атак. Системи виявлення вторгнень у мережу (NIDS) здатні підтримувати обробку DPI та захищати внутрішню мережу від зовнішніх атак. NIDS перевіряє заголовок пакета, покладається на методи зіставлення шаблонів для аналізу корисного набору пакетів і приймає рішення щодо значущості тіла пакета на основі вмісту корисного навантаження.

Завдання систем виявлення / запобігання вторгнень. Системи виявлення вторгнень (IDS) використовують кілька препроцесорів та механізм виявлення на основі правил, який виконує класифікацію пакетів та перевірку вмісту. IDS генерує оповіщення по пакету, і згодом кореляція між кількома оповіщеннями може вказувати на повний план атаки [16, 17, 18]. Правила IDS, такі як правила Snort [19] та Bleeding [20], що є програмним забезпеченням з відкритим кодом, складається з

частини, що відповідає заголовку, та частин, що відповідають корисному навантаженню. Перша частина перевіряє заголовок кожного вхідного пакета, використовуючи методи класифікації пакетів. Друга - аналізує корисне навантаження кожного пакета, виконуючи перевірку вмісту. Перевірка вмісту передбачає узгодження корисного набору пакетів із заздалегідь визначеними шаблонами, що описуються як статичні шаблони або регулярні вирази. Додаткові обмеження щодо розміщення вищезазначених зразків вносять додаткову складність у обробку завдань IDS.

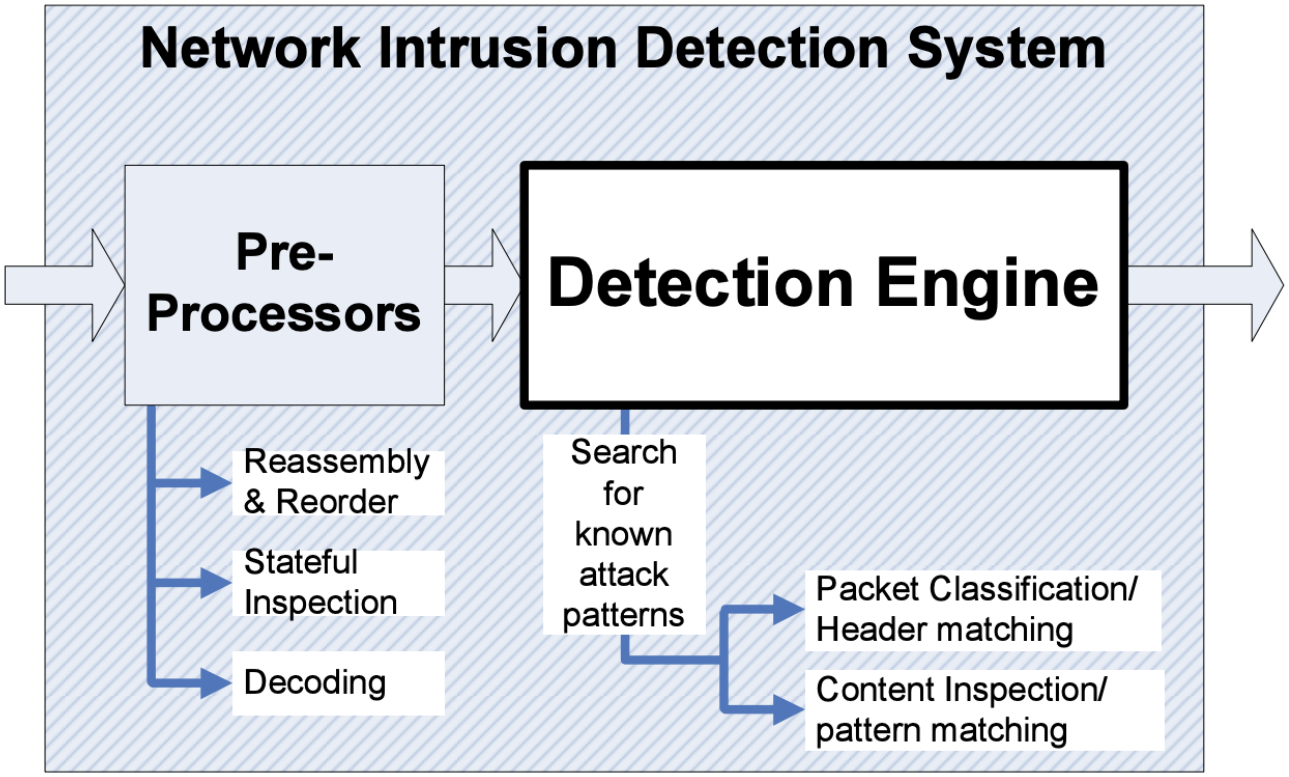

Рисунок 1.1 – Загальна схема систем IDS

Препроцесори: препроцесори IDS реалізують необхідні функції, які дозволяють механізму виявлення правильно перевіряти вхідний трафік за попередньо визначеними описами атак. Препроцесори відповідають за три види завдань. По-перше, вони збирають та переупорядковують TCP-пакети у більші. Це необхідно для виявлення атак, що охоплюють кілька пакетів. По-друге, вони виконують функції перевірки стану, такі як відстеження потоку або виявлення

портів; тобто, функції, пов'язані з рівнем протоколу, які відстежують різні зв'язки / потоки. Інспекція також може розглядатися як модуль який має огляд трафіку на рівні вище, ніж інспекція вмісту перевірка на наявність ненормальних подій, таких як переповнення буфера або атаки відмови в обслуговуванні (DoS). По-третє, препроцесори виконують спеціалізовані функції перевірки, переважно декодуючи різні види трафіку, наприклад, Telnet, FTP, RPC, HTTP, SMTP, пакети зі шкідливою закодованою інформацією. Після препроцесорів йде механізм виявлення, який використовує базу даних правил (набір правил) для опису шкідливих пакетів. Кожне правило має класифікацію пакетів та частину перевірки вмісту. Крім того, перевірка вмісту включає статичне узгодження шаблонів, відповідність регулярних виразів та обмеження розміщення шаблонів.

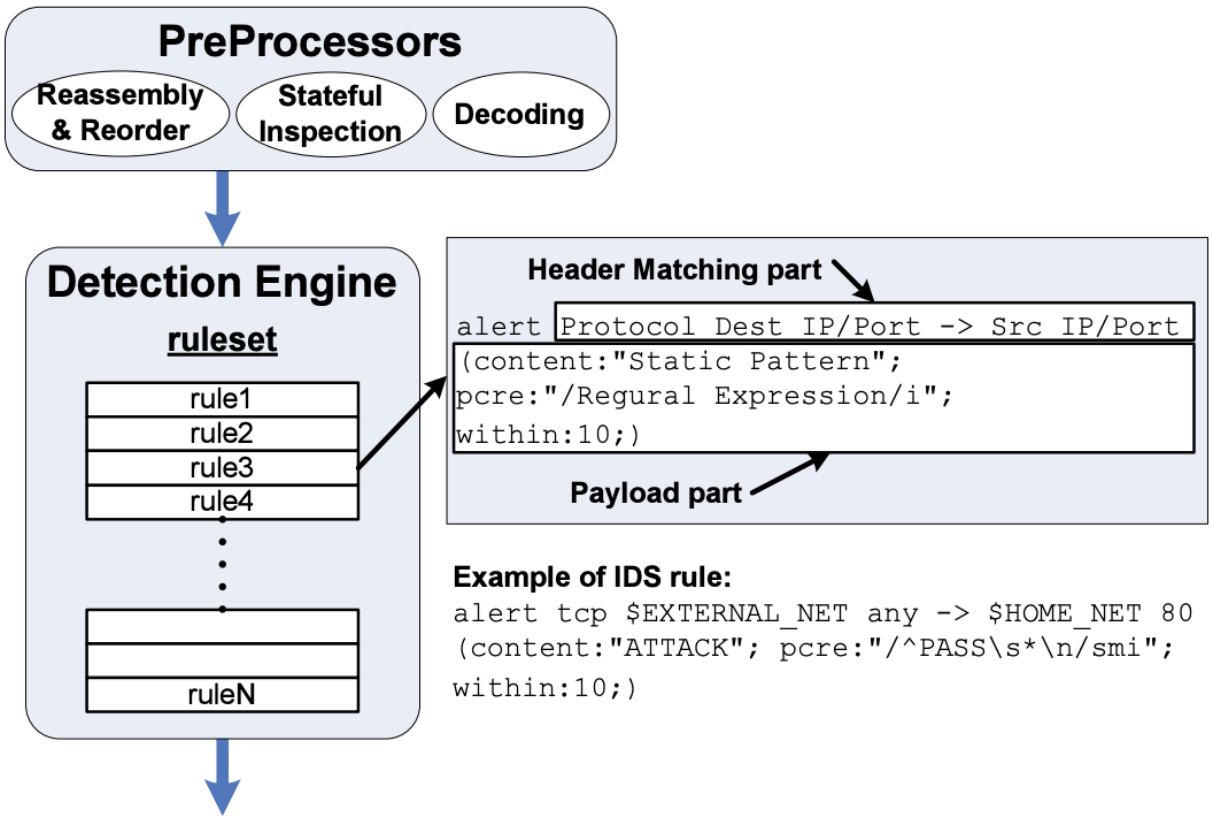

Рисунок 1.2 – Опис структури NIDS

Вхідний трафік спочатку перевіряється низкою препроцесорів, які виконують переупорядкування та повторну збірку пакетів, перевірку стану та декодування конкретних видів трафіку. Згодом пакети перевіряються на відповідність правилам,

які описують шкідливу діяльність. Кожне правило має заголовок пакета та опис корисного навантаження. У нижній правій частині рисунка - приклад правила IDS.

Класифікація пакетів: частина заголовка кожного правила NIDS описує заголовок потенційно небезпечного пакета. Опис заголовка може складатися з деяких або всіх наступних елементів:

- протокол;
- IP-адреса призначення;
- порт призначення;
- IP-адреса джерела;
- порт джерела;

Поля правил IP та портів можуть вказувати діапазони значень замість конкретної адреси або порту. Це робить класифікацію пакетів більш складною, ніж просто порівняння числових значень. У минулому багато дослідників пропонували різні методи класифікації пакетів та пошуку IP, такі як [21, 22, 23], тоді як деякі з них також використовують перенастроюване обладнання.

Статичне узгодження зразків: узгодження буквального значення заздалегідь визначених (статичних) зразків, є найважливішим завданням IDS. Статичні шаблони використовуються для опису шкідливого вмісту корисного навантаження та надання уявлення про дані пакетної програми. Правило IDS може містити один або кілька статичних шаблонів кількома байтами довжиною до декількох сотень байт. Статичний шаблон ATTACK позначається як зловмисний шаблон корисного навантаження, використовуючи вираз: content: "ATTACK". Паралельне збіг тисяч шаблонів корисного навантаження для вхідного пакету створює основні труднощі в роботі IDS. Спочатку системи IDS описували шкідливий вміст лише зі статичними шаблонами, проте нещодавно вони почали використовувати як статичні шаблони, так і регулярні вирази.

На даний момент IDS є найбільш ефективним рішенням для мережевої безпеки, що забезпечує обробку вмісту. Було обговорено найскладніші проблеми в роботі та реалізації IDS. Ядром IDS є механізм виявлення який використовує

великий набір правил описів атак. Основні функції – заголовок узгодження та узгодження корисного навантаження (перевірка вмісту).

Таблиця 1

Базовий синтаксис snort-PCRE;

Feature	**Description**
a	All ASCII characters, excluding meta-characters, match a single instance of themselves
[\^$.—?*+()	Meta-characters. Each one has a special meaning
.	Matches any character except “new line”
\?	Backslash escapes meta-characters, returning them to their literal meaning
[abc]	Character class. Matches one character inside the brackets. In this case, equivalent to (a|b|c)
[a-fA-F0-9]	Character class with range.
[^abc]	Negated character class. Matches every character except each non-Meta character inside breackets.
RegExp*	Kleene Star. Matches zero or more times the RegExpr.
RegExp+	Plus. Matches one or more times the RegExpr.
RegExp?	Question. Matches zero or one times the RegExpr.
RegExp{N}	Exactly. Matches N times the RegExpr.
RegExp{N, }	AtLeast. Matches N times or more the RegExpr.
RegExp{N,M}	Between. Matches N to M times the RegExpr.
\xFF	Matches the ASCII character with the numerical value indicated by the hexadecimal number FF.
\000	Matches the ASCII character with the numerical value indicated by the octal number 000.
\d, \w and \s	Shorthand character classes matching digits 0-9, word chars and whitespace, respectively.
\n, \r and \t	Match an LF char, CR char and a tab char, respectively.
(RegExp)	Groups RegExprs, so operators can be applied.
RegExp1RegExp2	Concatenation. RegExpr 1, followed by RegExpr 2.
RegExp1 | RegExp2	Union. RegExpr 1 or RegExpr 2.
^RegExp	Matches RegExpr only if at the beginning of the string.
RegExp$	Dollar. Matches RegExpr only if at the end of the string.
(?=RegExp), (?!RegExp), (?<=text), (?<!text)	Lookaround. Without consuming chars, stops the matching if the RegExp inside does not match.
(?(?=RegExp) then |else)	Conditional. If the lookahead succeeds, continues the matching with the “then” RegExp. If not, with the “else”RegExp.
\1, \2. . .\N	Backreferences. Have the same value as the text matchedby the corresponding pair of capturing parenthesis, from1st through Nth.
Flags	**Description**
i	Regular Expression becomes case insensitive.
s	Dot matches all characters, including newline.
m	^ and $ match after and before newlines.

Таблиця 2.

Функції у синтаксисі SNORT, які роблять розрахунки IDS більш важкими.

Feature	**Description**
depth	specifies how far into a packet Snort should search for the specified pattern.
offset	specifies where to start searching for a pattern within a packet.
distance	specifies how far into a packet Snort should ignore before starting to search for the specified pattern relative to the end of aprevious pattern match.
within	makes sure that at most *N* bytes are between pattern matches.
isdataat	verifies that the payload has data at a specific location, optionally looking if data relative to the end of the previous content match.
byte test	tests a byte field against a specific value (with operator i.e. less than (<), greater than (>), equal (=), not (!), bitwise AND (&), bitwise OR (ˆ) and various op- tions such as value, offset, relative, endian, string, and number type). Capable of testing binary values or converting representative byte strings to their binary equivalent and testing them.
byte jump	allows rules to be written for length encoded protocols. By having an option that reads the length of the portion of data,then skips that far forward in the packet, rules can be written that skip over specific portions of length-encoded protocolsand perform detection in very specific locations. Several op-tions are supported such as byte to convert, offset, relative, multiplier <value>, big/little endian, string, HEX/DEC/OCT, align and from beginning
dsize	tests the packet payload size.

Відповідність регулярним виразам: регулярні вирази використовуються при скануванні корисного набору пакетів IDS. Відкритий код IDS Snort and Bleeding Edge [19, 20], прийняв Perl-сумісний синтаксис регулярних виразів (PCRE) [24]. Приклад правила IDS на *Рис. 1.2* використовує вираз pcre: "/ ΛPASS \ s * \ n / smi"; для опису шкідливого вмісту у форматі регулярних виразів. Щоб ідентифікувати небезпечний пакет на основі цього правила, рядок, який відповідає регулярному виразу “/ ΛPASS \ s * \ n / smi”, повинен бути включений до вмісту пакету. Окрім добре відомих особливостей суворого визначення регулярних виразів, PCRE розширено новими операціями, такими як прапори та обмежені повторення. У *таблиці 1* описаний базовий синтаксис PCRE, що підтримується механізмами

узгодження шаблонів регулярних виразів. Узгодження регулярних виразів вважається значно ефективнішим і, водночас, більш складним та обчислювальним.

Розміщення шаблону та інші обмеження корисного навантаження: обмеження щодо корисного навантаження пакетів та розміщення шаблону корисного навантаження - це функції, які створюють додаткові труднощі при впровадженні IDS. У таблиці 2 представлені деякі функції синтаксису Snort, які ускладнюють правила IDS. Вищезазначені команди змінюють початкове значення частин правила вмісту, додаючи додаткові обмеження щодо розміщення відповідних шаблонів у корисному навантаженні пакета. Отже, правила можуть визначати корисну частину пакетів, де шаблон повинен відповідати узгодженому шаблону. Крім того, такі команди, як тестування байтів, вибирають і тестують поле корисного навантаження байта, використовуючи кілька числових або логічних операторів.

Кожне правило IDS може вказувати різні обмеження корисного навантаження для опису підозрілого пакета, використовуючи наведені в таблицях синтаксиси. Наприклад, правило IDS на Рис. 1.1 використовує твердження в межах: 10; заявити, що другий шаблон корисного навантаження (регулярний вираз / ΛPASS \ s * \ n / smi) повинен відповідати 10 байт після узгодження першого шаблону (ATTACK).

Вищезазначені обмеження створюють значні труднощі з реалізацією, змушуючи кожне правило вимагати окремого модуля (движок, потік тощо) для відстеження задоволених умов, вказівки частин корисного навантаження, які є дійсними для кожного зразка, щоб збігатися, і зберігання поле байтового навантаження, що перевіряється за допомогою команд перевірки байтів та переходу до байтів. Вищезазначені функції приносять значні витрати та обмежують продуктивність як програмних, так і апаратних реалізацій NIDS.

Висновки за розділом 1

У результаті аналізу науково-технічної літератури, було виділено методи пасивного мережевого аналізу, розглянуто механізми його роботи до складу яких входять SPI, MPI та DPI компоненти.

Проаналізовані системи безпеки, які використовують пасивний мережевий аналіз для ідентифікації загроз, а саме такі IDS/IPS як Snort та Bleeding. У таких мережах ідентифікація загроз відбувається за чітко прописаними правилами або ж шаблонами, які системи IDS/IPS беруть в обробку та по ним виявляють потенційні загрози.

РОЗДІЛ 2
МЕРЕЖЕВІ ЗАГРОЗИ ТА МЕТОДИ ЇХ КЛАСИФІКАЦІЇ

Вразливості у мережі виникають внаслідок слабких місць у проектуванні, конфігурації або реалізації комп'ютерних систем або мереж, що робить їх сприйнятливими до загроз. Загрози можуть виникнути в результаті використання недоліків конструкції, апаратного та програмного забезпечення, що використовується для побудови комп'ютерних мережевих систем. Системи можуть бути неправильно налаштовані і, отже, вразливі до атак. Такі вразливості можуть виникати через недосвідченість або недостатню підготовку відповідального персоналу, або незакінчену роботу. Вразливості також виникають через погане управління мережевими системами, наприклад, відсутність адекватних політик та процедур, які вказують, хто має доступ до яких ресурсів і коли, а також недостатня або рідкісна перевірка трафіку, що відбувається в мережевих системах.

2.1 Алгоритм класифікації Adaboost

Алгоритм AdaBoost, скорочене від Adaptive Boosting, - це техніка покращення, яка використовується подібно методу ансамблю в машинному навчанні. Така техніка називається Adaptive Boosting, оскільки ваги перепризначаються кожному екземпляру, а більш високі ваги призначають неправильно класифікованим екземплярам. Підсилення використовується для зменшення упередженості, а також дисперсії, для навчання під контролем. Це працює за принципом послідовного вирощування учнів. За винятком першого, кожен наступний студент вирощується з попередніх дорослих учнів. Простими словами, слабкі учні перетворюються на сильних.

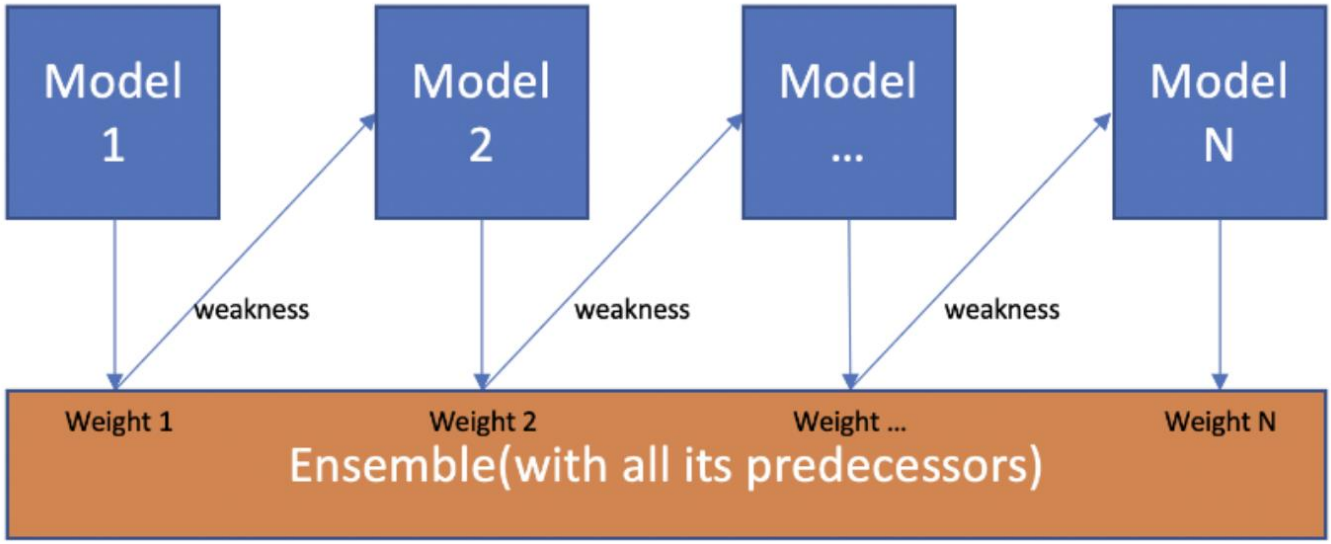

Рисунок 2.1 – Моделі 1-n це індивідуальні моделі (дерева рішень)

Малюнок показує, що коли зроблена перша модель і помилки з першої моделі зазначаються алгоритмом, запис, який неправильно класифікується, подається як вхід для наступної моделі. Цей процес повторюється доти, доки не буде дотримано вказану умову. Як можемо бачити на малюнку, існує N кількість моделей, зроблених, беручи помилки з попередньої моделі. Саме так працює підсилення. Моделі 1,2, 3,…, N - це окремі моделі, які називають деревами рішень. Усі типи підсилювальних моделей працюють за однаковим принципом.

Коли використовується випадковий ліс, алгоритм робить n кількість дерев. Це робить належні дерева, які складаються із стартового вузла з декількома листяними вузлами. Деякі дерева можуть бути більшими за інші, але в випадковому лісі немає фіксованої глибини. Але з Adaboost це не так. В AdaBoost алгоритм робить вузол лише з двома стулками, і це відоме як Stump (пень).

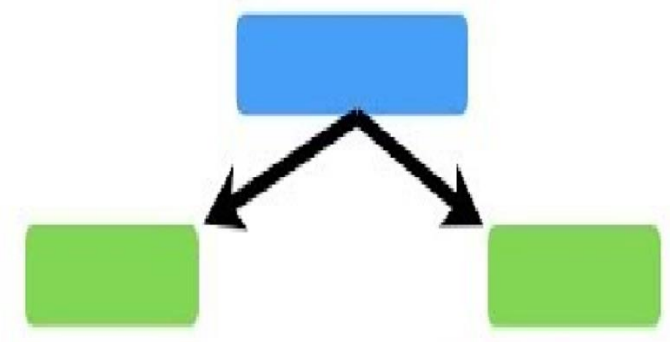

Рисунок 2.2 – Stump (пень)

Малюнок вище представляє пень. Чітко видно, що він має лише один вузол із лише двома стулками. Ці пні слабко навчаються, і прийоми посилення хотять цього. Порядок пнів дуже важливий в AdaBoost. Помилка першого пня впливає на те, як виготовляється інший пень.

Створення першого базового учня

Тепер настав час створити першого базового учня. Алгоритм приймає першу функцію, тобто функцію 1, і створює перший пень f1. З усіх цих пнів він створить три дерева прийняття рішень, і їх можна назвати базовими моделями пнів для учнів. З цих 3 моделей алгоритм вибирає лише одну. Для вибору базового учня є дві властивості: Джині та Ентропія. Ми повинні обчислити джинні або ентропію так само, як це розраховується для дерев рішень. Пень, який має найменше значення, буде першим базовим учнем. На малюнку нижче всі 3 пні можна зробити з 3-ма ознаками. Цифра під листками представляє правильно та неправильно класифіковані записи. За допомогою цих записів обчислюється індекс Джині або ентропії. Пень, що має найменшу ентропію, або Джині буде обраний для базового учня.

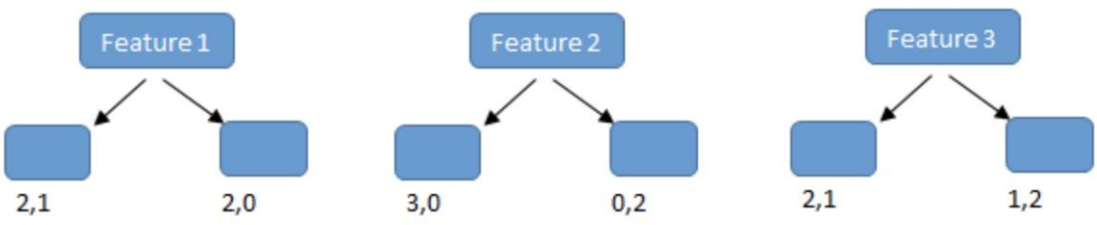

Рисунок 2.3 – Базові учні/пні

Розрахунок загальної помилки (TE):

Загальна похибка - це сума всіх помилок у секретному записі помножена на вагу вибірки.

$$TE = k * M,$$

де TE – загальна помилка,

k – кількість помилок,

M – вага вибірки

Розрахунок продуктивності пня:

$$\text{Продуктивність} = \frac{1}{2}\ln\frac{1-TE}{TE}$$

де TE – загальна помилка,

Поставивши значення загальної похибки у наведену вище формулу і після розв'язання, ми отримаємо значення продуктивності. Потрібно розрахувати потрібно розраховувати TE та продуктивність пня, щоб оновити вагу зразка, перш ніж переходити до наступної моделі або етапу, тому що, якщо застосовується однакова вага, ми отримуємо результат з першої моделі. Під час посилення лише неправильні записи / неправильно класифіковані записи отримують більше переваги, ніж правильно класифіковані записи. Таким чином, лише неправильні записи з дерева рішень / пні передаються іншому пню.

Оновлення ваг:

Для неправильно класифікованих записів використовується формула:

$$\text{Вага нового зразка} = \text{вага зразка} \times e^{\text{продуктивність}}$$

А для правильно класифікованих записів ми використовуємо ту саму формулу з від'ємним знаком із показником ефективності, так що вага для правильно класифікованих записів зменшиться порівняно з неправильно класифікованими. Формула:

$$\text{Вага нового зразка} = \text{вага зразка} \times e^{-\text{продуктивність}}$$

Створення нового набору даних

Наступним кроком буде створення нового набору даних із попереднього. У новому наборі даних частота неправильно класифікованих записів буде більшою, ніж правильна. Розглядаючи ці нормовані ваги, ми повинні створити новий набір даних, і цей набір даних базується на нормованих вагах. Ймовірно, він відбере неправильні записи для навчальних цілей. Це буде друге дерево рішень [25].

2.2 Алгоритм класифікації RandomForest

Випадковий ліс - це гнучкий, простий у використанні алгоритм машинного навчання, який, навіть без налаштування гіперпараметрів, більшу частину часу дає чудові результати. Це також один з найбільш часто використовуваних алгоритмів, завдяки своїй простоті та різноманітності (його можна використовувати як для класифікації, так і для регресії). У цій публікації ми дізнаємося, як працює алгоритм випадкового лісу, чим він відрізняється від інших алгоритмів та як ним користуватися.

Випадковий ліс - це керований алгоритм навчання. «Ліс», який він будує, - це ансамбль дерев-рішень, які зазвичай навчають методом «мішків». Загальна ідея методу мішків полягає в тому, що поєднання моделей навчання підвищує загальний результат.

Простіше кажучи: випадковий ліс будує кілька дерев рішень та об’єднує їх, щоб отримати більш точний та стабільний прогноз.

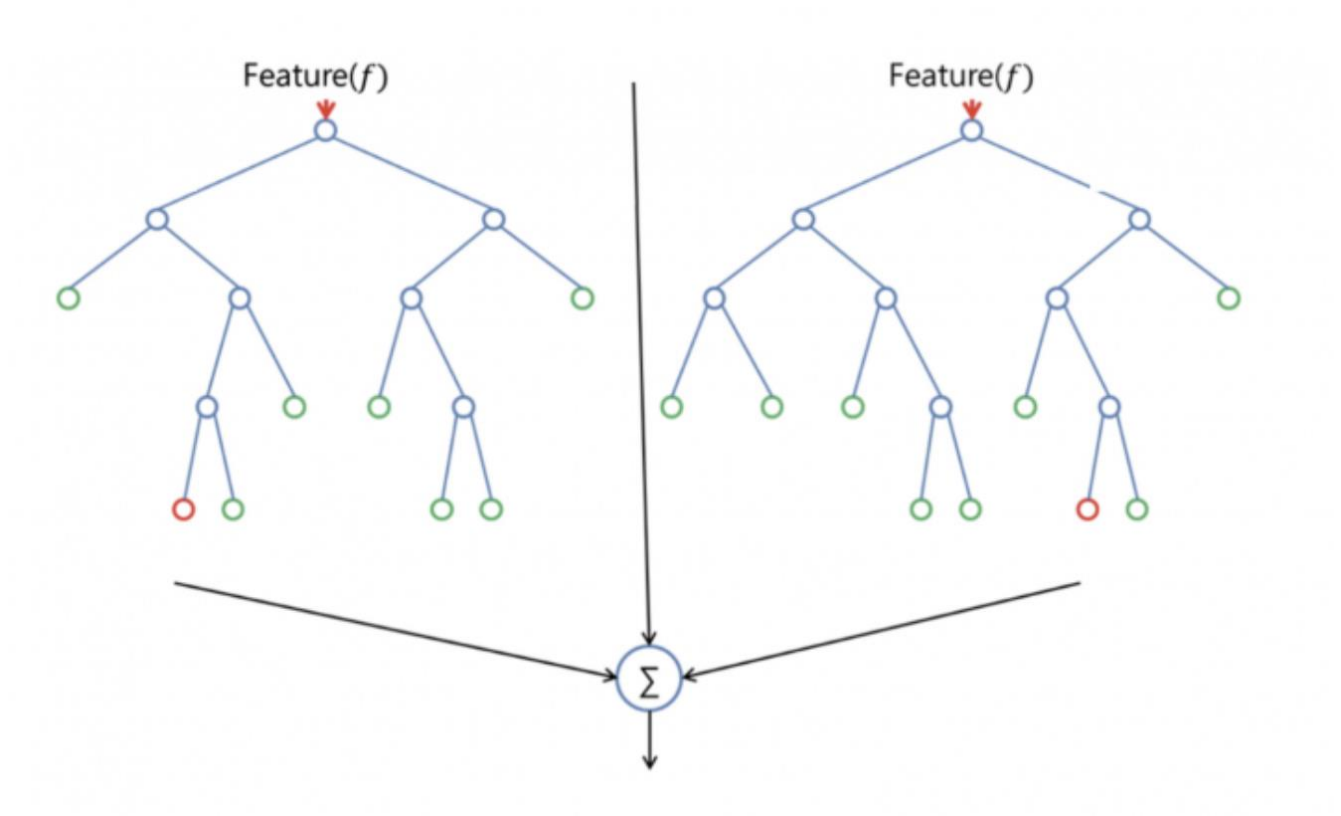

Рисунок 2.4 – Випадковий ліс з 2-ма деревами.

Випадковий ліс додає моделі додаткової випадковості під час вирощування дерев. Замість того, щоб шукати найважливішу функцію під час розбиття вузла, вона шукає найкращу функцію серед випадкової підмножини функцій. Це призводить до широкого розмаїття, що в цілому призводить до кращої моделі.

Отже, у випадковому лісі алгоритмом розбиття вузла враховується лише випадкова підмножина функцій. Ви навіть можете зробити дерева більш випадковими, додатково використовуючи випадкові пороги для кожної функції, а не шукаючи найкращі можливі пороги (як це робить звичайне дерево рішень).

Ще однією чудовою якістю алгоритму випадкового лісу є те, що дуже легко виміряти відносну важливість кожної ознаки для передбачення. Sklearn пропонує чудовий інструмент для цього, який вимірює важливість функції, розглядаючи, наскільки вузли дерев, що використовують цю функцію, зменшують домішки серед

усіх дерев у лісі. Він автоматично обчислює цей бал для кожної функції після тренування і масштабує результати, щоб сума всіх значень дорівнювала одиниці.

У дереві рішень кожен внутрішній вузол являє собою "тест" на атрибут (наприклад, перевертання монети піднімаються головки чи хвости), кожна гілка представляє результат тесту, а кожен листовий вузол представляє мітку класу (рішення, прийняте після обчислення всіх атрибутів). Вузол, який не має дочірніх елементів, є листом.

Подивившись на важливість функції, можна вирішити, які функції можливо викинути, оскільки вони недостатньо (або іноді взагалі нічого не вносять) у процес прогнозування. Це важливо, оскільки загальним правилом машинного навчання є те, що чим більше у вас функцій, тим більша ймовірність, що ваша модель постраждає від переобладнання та навпаки [26].

2.3 Алгоритм класифікації ExtraTrees

ExtraTreesClassifier - це ансамблевий метод навчання, заснований на деревах рішень. Класифікатор ExtraTreesClassifier, як і RandomForest, рандомізує певні рішення та підмножини даних, щоб мінімізувати надмірне вивчення даних та переобладнання.

Extra Trees схожий на випадковий ліс тим, що він будує декілька дерев та розбиває вузли, використовуючи випадкові підмножини функцій, але з двома ключовими відмінностями: він не завантажує спостереження (мається на увазі вибірки без заміни), а вузли діляться на випадкові поділи, а не найкращі розколи.
У додаткових деревах випадковість походить не від завантаження даних, а навпаки, від випадкових поділів усіх спостережень.
ExtraTrees названо так, тому що, по суті, це надзвичайно рандомізовані дерева. [27]

2.4 Алгоритм класифікації GradientBoost

Класифікатори посилення градієнта - це метод AdaBoosting у поєднанні із зваженою мінімізацією, після чого класифікатори та зважені вхідні дані перераховуються. Метою класифікаторів Gradient Boosting є мінімізація втрат або різниці між фактичним значенням класу навчального прикладу та прогнозованим значенням класу. Не потрібно розуміти процес зменшення втрат класифікатора, але він працює подібно до градієнтного спуску в нейронній мережі.

У випадку з машинами, що підсилюють градієнт, кожного разу, коли до моделі додається новий слабкий учень, ваги попередніх учнів заморожуються на місці, залишаючись незмінними при введенні нових шарів. Це відрізняється від підходів, що використовуються в AdaBoosting, де значення коригуються при додаванні нових учнів.

Потужність машин, що підсилюють градієнт, походить від того, що їх можна використовувати в більшості задач, ніж двійкові класифікації, вони можуть бути використані в задачах класифікації з кількома класами.

Для того, щоб реалізувати класифікатор посилення градієнта, нам потрібно виконати ряд різних кроків:

- Встановлення моделі
- Налаштування параметрів моделі
- Прогнозування
- Інтерпретування результатів

Встановлення моделей досить просто, оскільки нам, як правило, просто потрібно викликати команду fit () після налаштування моделі.

Однак налаштування гіперпараметрів моделі вимагає певного активного прийняття рішень. Існують різні аргументи, які ми можемо налаштувати, щоб спробувати отримати найкращу точність для моделі. Одним із способів ми можемо це зробити, змінивши швидкість навчання моделі. Ми хочемо перевірити ефективність моделі на навчальному наборі за різних темпів навчання, а потім використати найкращу швидкість навчання для прогнозування.

Прогнози можна зробити дуже просто, використовуючи функцію predict () після встановлення класифікатора. Вам потрібно буде передбачити особливості набору даних для тестування, а потім порівняти прогнози з фактичними мітками. Процес оцінки класифікатора, як правило, включає перевірку точності класифікатора, а потім налаштування параметрів / гіперпараметрів моделі, поки класифікатор не отримає точність, якою користувач задоволений. [28]

2.5 Алгоритм класифікації MPL (Multilayer Perceptron)

Персептрон є простим алгоритмом, призначеним для виконання двійкової класифікації; тобто він передбачає, чи належить вхід до певної категорії чи ні.

Персептрон - це лінійний класифікатор; тобто це алгоритм, який класифікує введення, розділяючи дві категорії прямою лінією. Вхідні дані - це, як правило, вектор ознак x, помножений на ваги w і доданий до зміщення b: y = w * x + b.

Персептрон виробляє одиничний вихід на основі кількох дійсних входів, формуючи лінійну комбінацію, використовуючи свої вхідні ваги.

Багатошаровий персептрон (MLP) - це глибока, штучна нейронна мережа. Він складається з більш ніж одного персептрона. Вони складаються з вхідного рівня для прийому сигналу, вихідного рівня, який приймає рішення або передбачення щодо вхідного сигналу, а між цими двома - довільної кількості прихованих шарів, які є справжнім обчислювальним механізмом MLP.

Багатошарові перцептрони часто застосовуються до контрольованих навчальних проблем вони тренуються на наборі пар input-output і вчаться моделювати кореляцію між цими входами та результатами. Навчання передбачає коригування параметрів, ваг та упереджень моделі, щоб мінімізувати помилки.

Зворотне розповсюдження використовується для здійснення цих коригувань зважування та зміщення щодо похибки, а саму похибку можна виміряти різними способами, включаючи середньоквадратичну похибку (RMSE).

Мережі прямого зв'язку, такі як MLP в основному беруть участь у двох рухах, постійному вперед і назад. Можемо сприймати цей пінг-понг здогадок та відповідей

як свого роду прискорену науку, оскільки кожна здогадка - це перевірка того, що, ми знаємо, і кожна відповідь - це зворотний зв'язок, що дає нам зрозуміти, наскільки ми помиляємось.

У прямому проході потік сигналу рухається від вхідного шару через приховані шари до вихідного шару, і рішення вихідного шару вимірюється щодо основних міток істини (вчителя).

У зворотному проході, використовуючи зворотне розповсюдження та ланцюгове правило числення, часткові похідні функції помилки w.r.t. різні ваги та упередження повторно поширюються через MLP. Цей акт диференціації дає нам градієнт або ландшафт помилок, уздовж якого параметри можуть коригуватися, коли вони рухають MLP на крок ближче до мінімуму помилок. Це можна зробити за допомогою будь-якого алгоритму оптимізації на основі градієнта, такого як стохастичний градієнтний спуск. [29]

Висновки за розділом 2

У ході написання другого розділу було виконані наступні завдання:

1. Проаналізовано алгоритму AdaBoost, що є мета-алгоритмом статистичної класифікації. Цей алгоритм можна використовувати разом з багатьма іншими типами алгоритмів навчання для підвищення продуктивності. Результат роботи інших алгоритмів навчання ("слабкі учні") об'єднується у зважену суму, яка представляє остаточний результат посиленого класифікатора. AdaBoost є адаптивним у тому сенсі, що наступні слабкі учні підлаштовуються на користь тих випадків, які неправильно класифікували попередні класифікатори.

2. Проаналізовано алгоритму RandomForest. Данний алгоритм є ансамблевим методом навчання для класифікації, регресії та інших завдань, який діє шляхом побудови безлічі дерев рішень під час навчання та виведення класу, або середнім / середнім прогнозуванням (регресія) окремих дерев.

3. Проаналізовано алгоритму ExtraTrees. Цей алгоритм є типом ансамблевої методики навчання, яка узагальнює результати декількох декорельованих дерев рішень, зібраних у «лісі», для виведення результату класифікації.

4. Проаналізовано алгоритму GradientBoost. Він є технікою машинного навчання для регресії, класифікації та інших завдань, яка створює модель прогнозування у вигляді ансамблю слабких моделей прогнозування, як правило, дерев рішень.

5. Проаналізовано алгоритму MPL. Цей алгоритм є класом штучної нейронної мережі прямого зв'язку. MLP складається щонайменше з трьох шарів вузлів: вхідного шару, прихованого шару та вихідного шару. За винятком вхідних вузлів, кожен вузол є нейроном, який використовує нелінійну функцію активації. MLP використовує контрольовану техніку навчання.

РОЗДІЛ 3
РІШЕННЯ ЗАДАЧІ КЛАСИФІКАЦІЇ НЕЙРОННИМИ МЕРЕЖАМИ

Комп’ютер, навіть якщо він є автономним, відстежує журнали багатьох дій, які користувачі виконують на ньому. Є записи користувачів, які входять у систему, і як довго вони входять в систему, а також є записи про оновлення системного або прикладного програмного забезпечення. Є багато інших подібних журналів. Мережевий комп’ютер веде облік багатьох додаткових дій. Наприклад, є записи про те, який користувач передав які файли до / з системи та коли, і хто від кого і коли отримував електронну пошту.

Пристрої, що полегшують та контролюють комунікаційні дії, можуть зберігати записи кожного біта інформації яка надходить до машини або залишає машину. В мережевих комп'ютерних системах є потенційно надзвичайно великі масиви інформації про трафік та журнали. Більшість видів діяльності є нормальними, але дуже мала частка видів діяльності може бути поза сферою звичного або очікуваного. Незвична діяльність можливо є потенційним вторгненням. Такі ненормальні дії можна знайти, зіставляючи журнали та набори даних про трафік із зразками відомих випадків ненормальних дій.

Однак набори даних про трафік та журнали потенційно дуже великі, неоднакові та постійно зростають. Моделі вторгнення можуть бути непростими та простими для пошуку. Тут ідеї машинного навчання можуть допомогти значною мірою допомогти у викритті потенційних моделей вторгнення та надати попередження керівництву мережі.

Нейронні мережі спрямовані на вилучення дійсних, нових, потенційно корисних та значущих шаблонів із набору даних, як правило, великого, у домені за допомогою нетривіального механізму навчання. Алгоритм машинного навчання намагається розпізнати складні шаблони в існуючих наборах даних, щоб допомогти приймати розумні рішення або передбачення, коли він стикається з новими або

раніше не баченими екземплярами даних. Фундаментальною проблемою, яку повинен вирішити алгоритм машинного навчання, є проблема узагальнення.

У практичній частині дипломної роботи було використано інтерактивну Python консоль Jupyter, що має можливості окремого построкового виконання коду. Це значно покращує робочий процес, коли деякі функції по тренуванню нейронних мереж-класифікаторів можуть займати досить великий час.

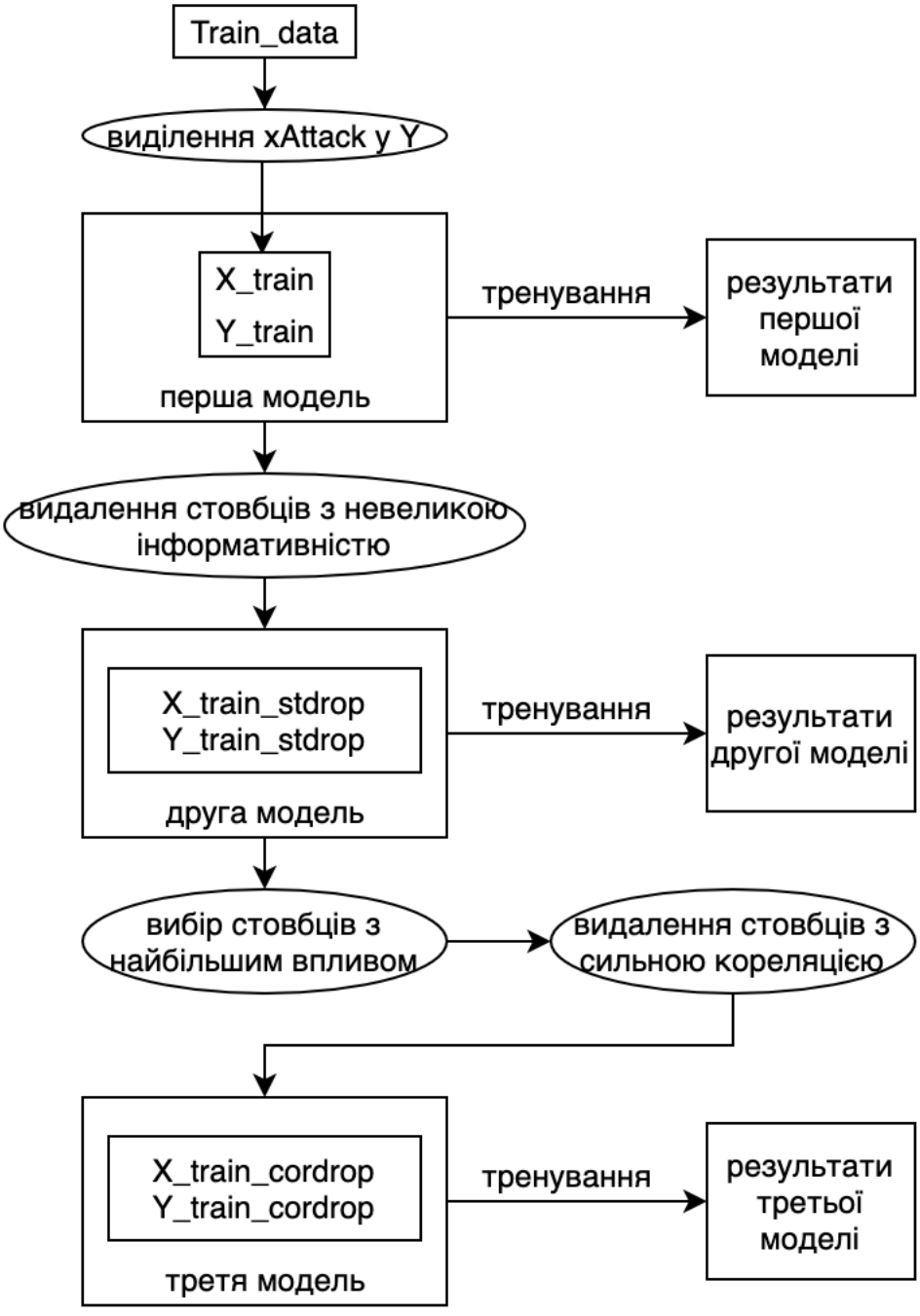

Рисунок 3.1 – Блок-схема процесу розробки моделей класифікації

Блок схема на Рис. 7 показує розроблений план дій по модифікації вхідних данних та формування з треннованих на їх основі алгоритмів різних моделей, якість яких буде порівняно.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib as matplot
import numpy as np

import re
import sklearn

import warnings
warnings.filterwarnings("ignore")

%matplotlib inline

df_train = pd.read_csv('./input/Train_data.csv')
df_test = pd.read_csv('./input/test_data.csv')
df_test = df_test.drop('Unnamed: 0',axis=1)
```

Рисунок 3.2 – Імпорт базових бібліотек

Початок розробки механізму ідентифікації загроз, використовуючи класифікатори на основі нейронних мереж починається з основних команд, що виконують імпорт базових бібліотек з використанням яких будуть виконані поставлені задачі.

Далі надається частини програмного коду з їх поясненням нижче:

import pandas as pd

Імпорт бібліотеки pandas за допомогою якої виконуються маніпуляції за датасетом та його модифікація.

import seaborn as sns

Імпорт бібліотеки для створення градієнтних графіків залежності (heatmap).

import matplotlib.pyplot as plt

Імпорт допоміжної бібліотеки для створення класичних графіків.

import matplotlib as matplot

Імпорт основної бібліотеки для створення класичних графіків.

import numpy as np

Імпорт бібліотелки з покращеними математичними операціями.

import re

import sklearn

Імпорт бібліотек для роботи з нейронними мережами з яких потім будуть імпортовані бібліотеки, що реалізують самі нейронні мережі навчання яких буде проводитись за наданими датасетами.

import warnings

warnings.filterwarnings("ignore")

Імпорт бібліотеки з попередженнями. Це потрібно для того щоб програмне середовище підтримувало методи для відображення помилок, якщо такі будуть. Для можливого подальшого виправлення та полегшення задачі налаштування.

%matplotlib inline

Параметр, що дозволяє ввід функцій по побудові графіків построково. Це дозволить більш компактно розміщати функції, та сприяє чистоті програмного коду.

df_train = pd.read_csv('./input/Train_data.csv')

df_test = pd.read_csv('./input/test_data.csv')

df_test = df_test.drop('Unnamed: 0',axis=1)

Імпорт датасетів “Train_data.csv” та “test_data.csv” та виключення ненайменованого столбця “Unnamed” у датасеті “df_test”.

```
df_train.head()
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate
0	0	icmp	20	2	491	0	0	0	0	0	...	25	0.17
1	0	udp	45	2	146	0	0	0	0	0	...	1	0.00
2	0	icmp	50	4	0	0	0	0	0	0	...	26	0.10
3	0	icmp	25	2	232	8153	0	0	0	0	...	255	1.00
4	0	icmp	25	2	199	420	0	0	0	0	...	255	1.00

dst_host_serror_rate	dst_host_srv_serror_rate	dst_host_rerror_rate	dst_host_srv_rerror_rate	xAttack
0.00	0.00	0.05	0.00	normal
0.00	0.00	0.00	0.00	normal
1.00	1.00	0.00	0.00	dos
0.03	0.01	0.00	0.01	normal
0.00	0.00	0.00	0.00	normal

Рисунок 3.3 – Вигляд імпортованого датасету з різноманітними полями, та «вчителем» - стовбець “xAttack”, що показує тип атаки, або ж надпис про те, що трафік нормальний.

Функція .head() повертає перші n рядків об'єкта на основі позиції. Це корисно для швидкого тестування, якщо у вашому об’єкті є потрібний тип даних.

Для від’ємних значень n така функція повертає всі рядки, окрім останніх n рядків, еквівалентно Python виразу df [: - n].

Параметри .head(): N, за числовим типом данних int, за замовчуванням це число становить 5.

N відповідає за кількість рядків для вибору.

[30]

```
df_test["xAttack"].value_counts()
```

```
normal    4329
dos       3332
r2l       1199
probe     1053
u2r         87
Name: xAttack, dtype: int64
```

Рисунок 3.4 – Функція .value_counts()

Функція .value_counts() підраховує усі унікальні значення стовбця та виводить їх статистику по кількості. Таким чином можемо побачити, що у стовбці xAttack існує 5 різних варіантів трафіку, що включають в себе типи атак (dos, r2l, probe, u2r) та нормальний трафік (normal). [31]

```
X_train = df_train.drop('xAttack', axis=1)
Y_train = df_train.loc[:,['xAttack']]
X_test = df_test.drop('xAttack', axis=1)
Y_test = df_test.loc[:,['xAttack']]
```

```
X_train.head(10)
```

_diff_srv_rate	dst_host_same_src_port_rate	dst_host_srv_diff_host_rate	dst_host_serror_rate	dst_host_srv_serror_rate	dst_host_rerror_rate	dst_host_srv_rerror_rate
0.03	0.17	0.00	0.00	0.00	0.05	0.00
0.60	0.88	0.00	0.00	0.00	0.00	0.00
0.05	0.00	0.00	1.00	1.00	0.00	0.00
0.00	0.03	0.04	0.03	0.01	0.00	0.01
0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.07	0.00	0.00	0.00	0.00	1.00	1.00
0.05	0.00	0.00	1.00	1.00	0.00	0.00
0.07	0.00	0.00	1.00	1.00	0.00	0.00
0.05	0.00	0.00	1.00	1.00	0.00	0.00
0.06	0.00	0.00	1.00	1.00	0.00	0.00

Рисунок 3.5 – Вивід функції .head(10)

Функція .drop видаляє рядки або стовпці, вказавши імена полів та відповідну вісь, або вказавши безпосередньо імена індексів або стовпців. При використанні мультиіндексної індексації поля на різних рівнях можна видалити, вказавши рівень.

Функція .loc дає доступ до групи рядків і стовпців за мітками або булевим масивом.

.loc [] в основному базується на мітках, але може також використовуватися з булевим масивом.

На рис. 11 функції .drop та .loc з бібліотеки pandas видаляють вищеописанний стовбець xAttack та виносить його в окремий датасет з одним стовбцем – xAttack. Надалі він буде використаний при тренуванні нейронних мереж та аналізі їх ефективності.

Як видно на рис таблиця датасету не має стовбця xAttack, що означає успішне використання функції .drop() для модифікування датасету. А також його виділення функцією .loc().

```
Y_train.head(10)
```

	xAttack
0	normal
1	normal
2	dos
3	normal
4	normal
5	dos
6	dos
7	dos
8	dos
9	dos

Рисунок. `3.6 – Вивід перших 10ти рядків таблиці Y_train функцією .head. Вона містить стовбець xAttack.

```
from sklearn import preprocessing
from sklearn.preprocessing import OneHotEncoder
```

```
le = preprocessing.LabelEncoder()
enc = OneHotEncoder()
lb = preprocessing.LabelBinarizer()
```

```
X_train["protocol_type"].value_counts()
```

```
icmp    102689
udp      14993
tcp       8291
Name: protocol_type, dtype: int64
```

Рисунок. 3.7 – Імпорт бібліотек препроцесінгу

from sklearn import preprocessing

sklearn - один із найбільш широко використовуваних пакетів Python для машинного навчання. Вона дозволяє виконувати операції та надає безліч алгоритмів. sklearn також пропонує відмінну документацію про свої класи, методи, функції, а також опис використовуваних алгоритмів.

sklearn підтримує:

- попередню обробку даних;

- зменшення розмірності;
- вибір моделей;
- регресії;
- класифікації;
- кластерний аналіз;

Він також пропонує кілька наборів даних, які можна використовувати для тестування моделей [32].

from sklearn.preprocessing import OneHotEncoder

Імпорт функцій по препроцессінгу, які перетврюють кожне унікальне появлення тексту у датасеті на його альтернативу представлену у числовому вигляді. Це дозволяє конвертацію данних та

le = preprocessing.LabelEncoder()
enc = OneHotEncoder()
lb = preprocessing.LabelBinarizer()

Введення скорочень для функцій препроцессінгу. Які таким чином спрощують використання функцій та візуальне сприйняття програмного коду.

X_train["protocol_type"].value_counts()

Виведення типів значень у колонці protocol_type за допомогою функції value_counts().

```
X_train['protocol_type'] = le.fit_transform(X_train['protocol_type'])

X_test['protocol_type'] = le.fit_transform(X_test['protocol_type'])

X_train.head(10)
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_count
0	0	0	20	2	491	0	0	0	0	0	...	150
1	0	2	45	2	146	0	0	0	0	0	...	255
2	0	0	50	4	0	0	0	0	0	0	...	255
3	0	0	25	2	232	8153	0	0	0	0	...	30
4	0	0	25	2	199	420	0	0	0	0	...	255
5	0	0	50	1	0	0	0	0	0	0	...	255
6	0	0	50	4	0	0	0	0	0	0	...	255
7	0	0	50	4	0	0	0	0	0	0	...	255
8	0	0	52	4	0	0	0	0	0	0	...	255
9	0	0	50	4	0	0	0	0	0	0	...	255

Рисунок. 3.8 – Процесс перетворення та вигляд датасету після нього. Стовбець protocol_type перетворений у цифрові значення.

Такі перетворення потрібні для того, щоб дозволити нейронним мережам класифікаторам коректно обробляти інформацію, адже вони не мають можливості обробляти значення стовбців у вигляді тексту. Тому такі значення будуть автоматично переведені у числовий вигляд так, як це виджно на рисунку 15.

```
X_train["protocol_type"].value_counts()
```

```
0    102689
2     14993
1      8291
Name: protocol_type, dtype: int64
```

Рисунок 3.9 – Вигляд перетворених значень у стовбці protocol_type.

Після таких перетворень проведемо аналогічні з стовбцями Y_train та Y_test:

```
Y_train['xAttack'] = le.fit_transform(Y_train['xAttack'])
lb.fit_transform(Y_train['xAttack'])

Y_test['xAttack'] = le.fit_transform(Y_test['xAttack'])
lb.fit_transform(Y_test['xAttack'])

Y_train['xAttack'].value_counts()
```

```
1    67343
0    45927
2    11656
3      995
4       52
Name: xAttack, dtype: int64
```

Рисунок 3.10 – Аналогічне перетворення стовбця xAttack, що містить класс трафіку у числовий вигляд аналогічним способом – використовуючи функцію .fit_transform.

```
Y_train.describe()
```

	xAttack
count	125973.000000
mean	0.744985
std	0.653748
min	0.000000
25%	0.000000
50%	1.000000
75%	1.000000
max	4.000000

Рисунок 3.11 – Функція describe застосована до датасету Y_train

Pandas description () використовується для перегляду деяких основних статистичних деталей, таких як процентиль, середнє значення, стандартне значення тощо, кадру даних або ряду числових значень.

Описова статистика включає статистику, яка узагальнює центральну тенденцію, дисперсію та форму розподілу набору даних, виключаючи значення NaN.

Метод Pandas description () аналізує числові та об'єктні ряди, а також набори стовпців DataFrame змішаних типів даних. Результат буде змінюватися залежно від того, що надається. Докладніше див. У примітках нижче.

Вивід функції .describe() показує основні статистичні параметри.

Count – сумма усіх значень

Mean – середнє арифметичне

Std – дисперсія (Standart Deviation)

Min – мінімальне значення

25%, 50%, 75% – 25% значень Х або меньше, 50% – Х або меньше і тд.

Max – Максимальне значення серед усіх

Функція .std() обчислює середнє відхилення вздовж зазначеної осі.

Повертає стандартне відхилення, міру розповсюдження розподілу елементів масиву. Стандартне відхилення обчислюється для сплощеного масиву за замовчуванням, інакше за вказаною віссю [33].

Використання цієї функції знадобиться для наступних дій - визначення найменьших середніх значень серед стовбців та усунення їх, для покращення датасету шляхом видалення малоінформативних стовбців, які будуть характеризуватися маленьким середнім значенням по всьому датасету. Визначення таких стовбців відбувається за допомогою функції .std().

```
#except continuous feature
con_list = ['protocol_type', 'service', 'flag', 'land', 'logged_in', 'su_attempted', 'is_host_login',
con_train = X_train.drop(con_list, axis=1)

#drop n smallest std features
stdtrain = con_train.std(axis=0)
std_X_train = stdtrain.to_frame()
std_X_train.nsmallest(10, columns=0).head(10)
```

	0
num_outbound_cmds	0.000000
urgent	0.014366
num_shells	0.022181
root_shell	0.036603
num_failed_logins	0.045239
num_access_files	0.099370
dst_host_srv_diff_host_rate	0.112564
diff_srv_rate	0.180314
dst_host_diff_srv_rate	0.188922
wrong_fragment	0.253530

Рисунок 3.12 – Вивід 10-ти найменьших середніх значень по стовбцям.

Функція .nsmallest() повертає перші n рядків, упорядкованих за стовпцями у порядку зростання. Не вказані стовпці також повертаються, але не використовуються для упорядкування.

Цей метод еквівалентний df.sort_values (стовпці, зростаючий = True) .head (n), але більш ефективний. [34]

Перші 2 рядки коду видаляють деякі стовбці, числові значення яких є посиланням на щось, або просто не мають відношення до класифікації, та тільки будуть давати нерелевантні поля у процесс тренування. Таким чином ми присвоюємо такий покращенний датасет змінній con_train.

Далі з цього списку визначається середне арифметичне усіх значень та виводится 10 найменьших значень серед них. Як можна побачити на рисунку, середне арифметичне змінної num_outbound_cmds є нулем, тому ми надалі виключемо його з датасету.

```
X_train = X_train.drop(['num_outbound_cmds'], axis=1)
X_test = X_test.drop(['num_outbound_cmds'], axis=1)

df_train = pd.concat([X_train, Y_train], axis=1)
```

Рисунок 3.13 – Виключення стовбця num_outbound_cmds та присвоення змінній df_train з’єднаного датасету із X_train та X_test.

```
df_train.columns
```

```
Index(['duration', 'protocol_type', 'service', 'flag', 'src_bytes',
       'dst_bytes', 'land', 'wrong_fragment', 'urgent', 'hot',
       'num_failed_logins', 'logged_in', 'num_compromised', 'root_shell',
       'su_attempted', 'num_root', 'num_file_creations', 'num_shells',
       'num_access_files', 'is_host_login', 'is_guest_login', 'count',
       'srv_count', 'serror_rate', 'srv_serror_rate', 'rerror_rate',
       'srv_rerror_rate', 'same_srv_rate', 'diff_srv_rate',
       'srv_diff_host_rate', 'dst_host_count', 'dst_host_srv_count',
       'dst_host_same_srv_rate', 'dst_host_diff_srv_rate',
       'dst_host_same_src_port_rate', 'dst_host_srv_diff_host_rate',
       'dst_host_serror_rate', 'dst_host_srv_serror_rate',
       'dst_host_rerror_rate', 'dst_host_srv_rerror_rate', 'xAttack'],
      dtype='object')
```

Рисунок 3.14 – Стовбці датасету df_train виражені у вигляді масиву. З відсутністю значення num_outbound_cmds

Вигляд датасету df_train показує те, що поля 'urgent', 'num_shells', 'root_shell', 'num_failed_logins', 'num_access_files', 'dst_host_srv_diff_host_rate', 'diff_srv_rate', 'dst_host_diff_srv_rate' та 'wrong_fragment' присутні у ньому до модифікації.

```
stdrop_list = ['urgent', 'num_shells', 'root_shell',
        'num_failed_logins', 'num_access_files', 'dst_host_srv_diff_host_rate',
        'diff_srv_rate', 'dst_host_diff_srv_rate', 'wrong_fragment']

X_test_stdrop = X_test.drop(stdrop_list, axis=1)

X_train_stdrop = X_train.drop(stdrop_list, axis=1)

df_train_stdrop = pd.concat([X_train_stdrop, Y_train], axis=1)

df_train_stdrop.head()
```

Рисунок 3.15 – Видалення малоінформаційних стовбців функцією .drop()

Виділення стовбців з маленьким середнім значенням, які можна побачити на рисунку. Та назначення покарешеного датасету у змінні X_test_stdrop та X_train_stdrop а також назначення з’єднаного датасету у змінну df_train_stdrop.

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	hot	logged_in	num_compromised	...	srv_diff_host_rate	dst_host_count
0	0	0	20	2	491	0	0	0	0	0	...	0.00	150
1	0	2	45	2	146	0	0	0	0	0	...	0.00	255
2	0	0	50	4	0	0	0	0	0	0	...	0.00	255
3	0	0	25	2	232	8153	0	0	1	0	...	0.00	30
4	0	0	25	2	199	420	0	0	1	0	...	0.09	255

Рисунок 3.16 – Вигляд таблиці df_train_stdrop

Як можна побачити, у таблиці df_train_stdrop тепер відсутні поля 'urgent', 'num_shells', 'root_shell', 'num_failed_logins', 'num_access_files', 'dst_host_srv_diff_host_rate', 'diff_srv_rate', 'dst_host_diff_srv_rate' та 'wrong_fragment'. Що підвищує його інформативність.

```
from sklearn import linear_model
LR = linear_model.LinearRegression()
```

```
LR.fit(X_train, Y_train)
```

```
LinearRegression()
```

```
lr_score = LR.score(X_test, Y_test)
print('Linear regression Score: %.2f %%' % lr_score)
```

```
Linear regression Score: 0.33 %
```

Рисунок 3.17 – Тренування та розрахунок ефективності нейронної мережі на основі алгоритму лінійної регресії. Ефективність у 33% не є задовільним результатом.

Першою моделлю є модель заснована на алгоритмі лінійної регресії, що є базовою моделлю машиного навчання, яка по суті усереднює результат по всім стовбцям та видає такий самий усереднений результат. У такому випадку не є дивним, що така модель не здобула значних результатів та має ефективність всього 33%.

На рисунку 3.18 відбувається імпортування та назначення скорочень для методів класифікації, для полегшення подальшого використання. Це дозволить робити звернення до скороченого атрибуту.

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.tree import DecisionTreeClassifier

AB = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(), n_estimators=100, learning_rate=1.0)
RF = RandomForestClassifier(n_estimators=10, criterion='entropy', max_features='auto', bootstrap=True)
ET = ExtraTreesClassifier(n_estimators=10, criterion='gini', max_features='auto', bootstrap=False)
GB = GradientBoostingClassifier(loss='deviance', learning_rate=0.1, n_estimators=200, max_features='auto')
```

Рисунок 3.18 – Імпортування та назначення скорочень для методів класифікації

```
AB.fit(X_train, Y_train)
```

```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(), n_estimators=100)
```

Рисунок 3.19 – Функія, що виконує тренування по заданим аргументам X_train та Y_train

```
AB_feature = AB.feature_importances_
AB_feature
```

```
array([           nan,            nan,            nan, 1.53801873e-04,
                  nan,            nan, 4.91423222e-06, 1.13445664e-04,
       1.03133762e-06,            nan, 9.72885388e-06, 6.63565707e-04,
       1.61346320e-04, 1.73358767e-06, 1.36501505e-08,            nan,
       5.75837435e-06,            inf,            nan, 0.00000000e+00,
       4.19843048e-06,            nan,            nan,            nan,
       4.14042716e-06, 1.11148870e-04, 1.11680328e-03,            nan,
                  nan,            nan,            nan,            nan,
                  nan,            nan,            nan,            nan,
                  nan,            nan,            nan,            nan])
```

Рисунок 3.20 – Коофіцієнти (ваги) аргументів класифікатора AdaBoost

Коофіцієнти, що показує агрумент .feature_importances_ показує наскільки важливі коофіцієнти у класифікаторах.

```
AB_feature.shape, X_train.shape
```

```
((40,), (125973, 40))
```

Рисунок 3.21 – Форма масивів AB_feature (коофіцієнти) та X_train (датасет навчання)

Коофіцієнти AB_feature складаются з 40 стовбців, а X_train з 40 стовбців та близько 126-ти тисяч рядків. Це показує вигляд “розв’язку” задачі класифікації шляхом підбору

```
ab_score = AB.score(X_test, Y_test)
print('AdaBoostClassifier Score: %.3f %%' % ab_score)
```

```
AdaBoostClassifier Score: 0.755 %
```

Рисунок 3.22 – Розразунок ефективності класифікатора за алгоритмом AdaBoost

Ефективність кожного алгоритму розраховується за допомогою функції .score яка порівнює передбачення нейромережі навченої на основі датасету X_test з готовими результатами записаних у датасеті Y_test. Потім, порівнявши кількість правильних та неправильних передбаень вона видає відповідь у відсотках розраховуючи це значення з відношення правильних передбачень на загальну кількість ціх передбачень.

У данному випадку отримано ефективність передбачень 75,5%, що є доволі непоганим результатом.

```
RF.fit(X_train, Y_train)
```

```
RandomForestClassifier(criterion='entropy', n_estimators=10)
```

```
RF_feature = RF.feature_importances_

rf_score = RF.score(X_test, Y_test)

print('RandomForestClassifier Score: %.3f %%' % rf_score)
```

```
RandomForestClassifier Score: 0.759 %
```

Рисунок. 3.23 – Тренування та розразунок ефективності класифікатора виконаним за алгоритмом RandomForest

```
ET.fit(X_train, Y_train)
```

```
ExtraTreesClassifier(n_estimators=10)
```

```
et_score = ET.score(X_test, Y_test)
print('ExtraTreeClassifier: %.3f %%' % et_score)
```

```
ExtraTreeClassifier: 0.770 %
```

Рисунок. 3.24 – Тренування та розразунок ефективності класифікатора виконаним за алгоритмом RandomForest

```
ET.fit(X_train, Y_train)
```

```
ExtraTreesClassifier(n_estimators=10)
```

```
et_score = ET.score(X_test, Y_test)
print('ExtraTreeClassifier: %.3f %%' % et_score)
```

```
ExtraTreeClassifier: 0.770 %
```

```
GB.fit(X_train, Y_train)
```

```
GradientBoostingClassifier(max_features='auto', n_estimators=200)
```

```
GB_feature = GB.feature_importances_

gb_score = GB.score(X_test, Y_test)

print('GradientBoostingClassifier Score: %.3f %%' % gb_score)
```

```
GradientBoostingClassifier Score: 0.767 %
```

Рисунок. 3.25 – Тренування та розразунок ефективності класифікаторів виконаних за алгоритмами ExtraTrees та GradientBoost.

```
cols = X_train.columns.values

feature_df = pd.DataFrame({'features': cols,
                           'AdaBoost' : AB_feature,
                           'RandomForest' : RF_feature,
                           'ExtraTree' : ET_feature,
                           'GradientBoost' : GB_feature
                          })
feature_df.head(8)
```

	features	AdaBoost	RandomForest	ExtraTree	GradientBoost
0	duration	NaN	0.004014	0.002166	0.001526
1	protocol_type	NaN	0.018903	0.029672	0.011368
2	service	NaN	0.035917	0.024023	0.007861
3	flag	0.000154	0.038171	0.009691	0.001157
4	src_bytes	NaN	0.197457	0.022271	0.361478
5	dst_bytes	NaN	0.050856	0.006427	0.062364
6	land	0.000005	0.000041	0.000028	0.000036
7	wrong_fragment	0.000113	0.005427	0.016213	0.013347

Рисунок. 3.26 – Перші 8 коофіцієнтів кожного алгоритма напроти кожного значення столбців.

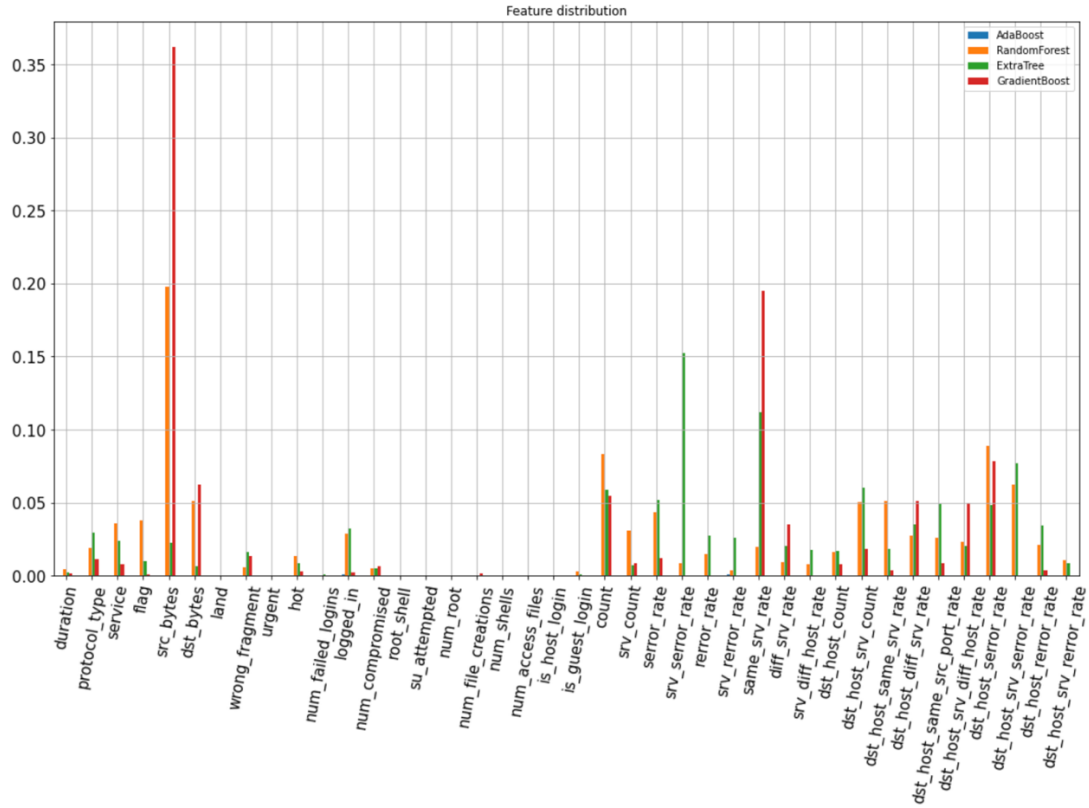

Рисунок 3.27 – Розподіл коофіцієнтів на кожному з алгоритмів показаних на графіку

На рисунку 3.28 можна побачити графік розподілу значимості підібраних коофіцієнтів (ваг). На основі цього потім буде сформовано окремий покращенний датасет, який буде містити тільки найвагоміші та найвпливовіші стовбці.

	features	AdaBoost	RandomForest	ExtraTree	GradientBoost
17	num_shells	inf	0.000085	0.000034	0.000455
26	srv_rerror_rate	0.001117	0.003892	0.025900	0.000019
11	logged_in	0.000664	0.029098	0.032084	0.002512
12	num_compromised	0.000161	0.005335	0.004944	0.006701
3	flag	0.000154	0.038171	0.009691	0.001157
7	wrong_fragment	0.000113	0.005427	0.016213	0.013347
25	rerror_rate	0.000111	0.014587	0.027322	0.000029
10	num_failed_logins	0.000010	0.000259	0.000634	0.000149
16	num_file_creations	0.000006	0.000143	0.000157	0.001475
6	land	0.000005	0.000041	0.000028	0.000036
20	is_guest_login	0.000004	0.003129	0.001153	0.000087
24	srv_serror_rate	0.000004	0.008359	0.152183	0.000349
27	same_srv_rate	NaN	0.019539	0.111794	0.194923
37	dst_host_srv_serror_rate	NaN	0.062133	0.077084	0.000317
31	dst_host_srv_count	NaN	0.050070	0.059961	0.017986
21	count	NaN	0.083173	0.058573	0.054406
23	serror_rate	NaN	0.043145	0.052124	0.012289
34	dst_host_same_src_port_rate	NaN	0.026073	0.049139	0.008559
36	dst_host_serror_rate	NaN	0.089068	0.048602	0.078094
33	dst_host_diff_srv_rate	NaN	0.027447	0.034780	0.050967
38	dst_host_rerror_rate	NaN	0.020956	0.034364	0.003468
1	protocol_type	NaN	0.018903	0.029672	0.011368
4	src_bytes	NaN	0.197457	0.022271	0.361478
5	dst_bytes	NaN	0.050856	0.006427	0.062364
35	dst_host_srv_diff_host_rate	NaN	0.022825	0.020470	0.049495
28	diff_srv_rate	NaN	0.009183	0.020595	0.035088
32	dst_host_same_srv_rate	NaN	0.051301	0.018593	0.003460
2	service	NaN	0.035917	0.024023	0.007861

Рисунок. 3.29 – Розподіл коофіцієнтів на кожному з алгоритмів показаних у вигляді таблиці.

```
selected_features = result['features'].values.tolist()
selected_features
```

```
['num_failed_logins',
 'logged_in',
 'dst_host_srv_rerror_rate',
 'num_compromised',
 'wrong_fragment',
 'is_guest_login',
 'num_file_creations',
 'num_shells',
 'num_root',
 'srv_rerror_rate',
 'urgent',
 'dst_host_serror_rate',
 'same_srv_rate',
 'dst_host_rerror_rate',
 'dst_host_srv_serror_rate',
 'srv_serror_rate',
 'protocol_type',
 'service',
 'src_bytes',
 'dst_host_diff_srv_rate',
 'count',
 'dst_bytes',
 'dst_host_srv_diff_host_rate',
 'diff_srv_rate',
 'dst_host_srv_count',
 'serror_rate',
 'dst_host_same_srv_rate',
 'dst_host_same_src_port_rate',
 'srv_count']
```

Рисунок. 3.30 – Список вибраних найбільш впливових стовбців.

Спираючись на таблицю вище було вибрано найбільш впливові коофіцієнти (ваги) для подальшого модифікування датасету, вибору впливових коофіцієнтів, видалення інших. Такі операції дозволять покращити датасети виключивши інформацію яка може негативно впливати на формування неправильних коофіцієнтів.

```
X_train_ens = X_train[selected_features]
X_train_ens.head()
```

	num_failed_logins	logged_in	dst_host_srv_rerror_rate	num_compromised	wrong_fragment	is_guest_login	num_file_creations	num_shells	num_root
0	0	0	0.00	0	0	0	0	0	0
1	0	0	0.00	0	0	0	0	0	0
2	0	0	0.00	0	0	0	0	0	0
3	0	1	0.01	0	0	0	0	0	0
4	0	1	0.00	0	0	0	0	0	0

5 rows × 29 columns

```
X_test_ens = X_test[selected_features]
X_test_ens.head()
```

	num_failed_logins	logged_in	dst_host_srv_rerror_rate	num_compromised	wrong_fragment	is_guest_login	num_file_creations	num_shells	num_root
0	0	0	0.32	0	0	0	0	0	0
1	0	1	0.00	0	0	0	0	0	0
2	0	1	0.00	0	0	0	0	0	0
3	0	1	0.00	0	0	0	0	0	0
4	0	0	0.00	0	0	0	0	0	0

5 rows × 29 columns

Рисунок. 3.31 – Формування датасетів вибраних із найбільш впливових стовбців та присвоення таких датасетів у змінні X_test_ens та X_train_ens.

Градієнтний графік на рисунку 3.32 дає можливість провести візуальний аналіз залежностей різних стовбців одне на одного.

Таким чином можна визначити стовбці з великою корреляцією, що, по суті, означає, що вони мають відомості про одні і ті ж самі речі. Отже, виключаючи високо корреляційні стовбці можливо ще трохи покращити датасет, та можливо підвищити ефективність моделей, навченних на цьому датасеті.

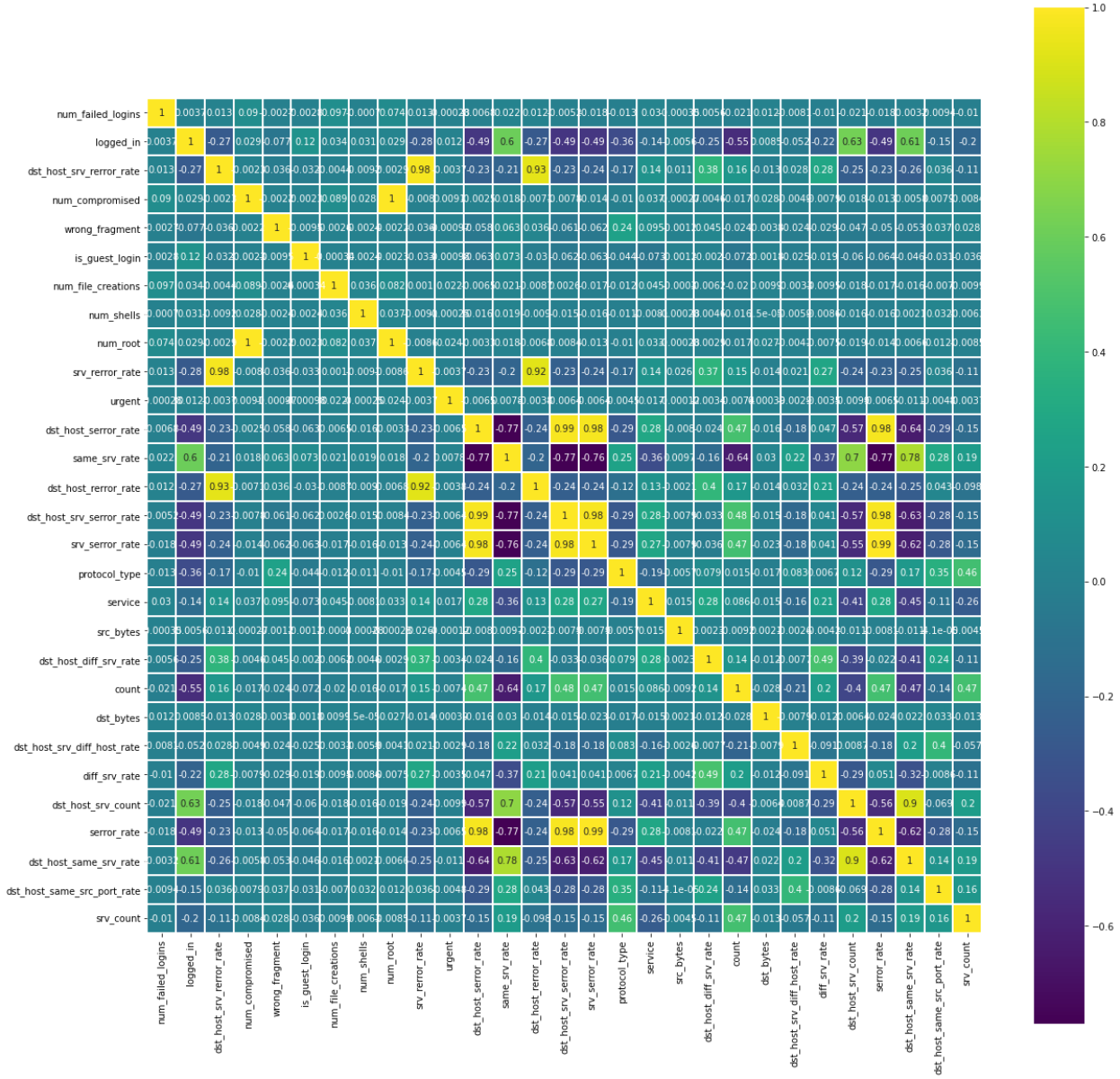

Рисунок 3.33 – Градієнтний графік залежностей коофіцієнтів.

На рисунку 3.34 виконуються команди по видаленню високо корреляційних стовбців використовуючи функцію .drop(), а перед цим вибравши їх та назначивши у змінну selected2.

```
selected2 = ['dst_host_serror_rate', 'serror_rate']
X_train_cordrop = X_train_ens.drop(selected2, axis=1)
X_train_cordrop.describe()
```

	rerror_rate	num_compromised	wrong_fragment	srv_rerror_rate	flag	srv_serror_rate	num_failed_logins	land	is_guest_login	num
count	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	
mean	0.119958	0.279250	0.022687	0.121183	2.575179	0.282485	0.001222	0.000198	0.009423	
std	0.320436	23.942042	0.253530	0.323647	1.141552	0.447022	0.045239	0.014086	0.096612	
min	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	2.000000	0.000000	0.000000	0.000000	0.000000	
50%	0.000000	0.000000	0.000000	0.000000	2.000000	0.000000	0.000000	0.000000	0.000000	
75%	0.000000	0.000000	0.000000	0.000000	4.000000	1.000000	0.000000	0.000000	0.000000	
max	1.000000	7479.000000	3.000000	1.000000	11.000000	1.000000	5.000000	1.000000	1.000000	

8 rows × 26 columns

```
X_test_cordrop = X_test_ens.drop(selected2, axis=1)
X_test_cordrop.describe()
```

	rerror_rate	num_compromised	wrong_fragment	srv_rerror_rate	flag	srv_serror_rate	num_failed_logins	land	is_guest_login	num_file
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.000000	10000.000000	10000.00000	10
mean	0.241316	0.020500	0.009000	0.238538	2.20290	0.101696	0.021200	0.000500	0.02770	
std	0.418059	0.289289	0.146701	0.418474	1.07854	0.295464	0.148838	0.022356	0.16412	
min	0.000000	0.000000	0.000000	0.000000	1.00000	0.000000	0.000000	0.000000	0.00000	
25%	0.000000	0.000000	0.000000	0.000000	2.00000	0.000000	0.000000	0.000000	0.00000	
50%	0.000000	0.000000	0.000000	0.000000	2.00000	0.000000	0.000000	0.000000	0.00000	
75%	0.330000	0.000000	0.000000	0.232500	2.00000	0.000000	0.000000	0.000000	0.00000	
max	1.000000	15.000000	3.000000	1.000000	11.00000	1.000000	3.000000	1.000000	1.00000	

Рисунок. 3.35 – Виключення високо залежних коофіцієнтів виходячи з візуального аналізу градієнтного графіку.

Після цього виаористано функцію .describe() для опису загальних параметрів різних полей датасету, водночас можна побачити, що полей які значаться у змінній selected2 не має у виводі функції .describe().

```
AB.fit(X_train_cordrop, Y_train)
```

```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(), n_estimators=100)
```

```
ab_finalscore = AB.score(X_test_cordrop, Y_test)
print('AdaBoostClassifier_final Score: %.3f %%' % ab_finalscore)
```

```
AdaBoostClassifier_final Score: 0.764 %
```

```
RF.fit(X_train_cordrop, Y_train)
```

```
RandomForestClassifier(criterion='entropy', n_estimators=10)
```

```
rf_finalscore = RF.score(X_test_cordrop, Y_test)
print('RandomForestClassifier_final Score: %.3f %%' % rf_finalscore)
```

```
RandomForestClassifier_final Score: 0.753 %
```

```
ET.fit(X_train_cordrop, Y_train)
```

```
ExtraTreesClassifier(n_estimators=10)
```

```
et_finalscore = ET.score(X_test_cordrop, Y_test)
print('ExtraTreesClassifier_final Score: %.3f %%' % et_finalscore)
```

```
ExtraTreesClassifier_final Score: 0.763 %
```

```
GB.fit(X_train_cordrop, Y_train)
```

```
GradientBoostingClassifier(max_features='auto', n_estimators=200)
```

```
gb_finalscore = GB.score(X_test_cordrop, Y_test)
print('GradientBoostClassifier_final Score: %.3f %%' % gb_finalscore)
```

```
GradientBoostClassifier_final Score: 0.766 %
```

```
LR.fit(X_train_cordrop, Y_train)
```

```
LinearRegression()
```

```
lr_finalscore = LR.score(X_test_cordrop, Y_test)
print('LinearRegression_final Score: %.3f %%' % lr_finalscore)
```

```
LinearRegression final Score: 0.339 %
```

Рисунок. 3.36 – Тренування та розразунок еффективності класифікаторів виконаних за алгоритмами AdaBoost, RandomForest, Linear regression, ExtraTrees та GradientBoost які використовували фінальну версію модифікованого датасету.

Після усіх модифікацій датасету проводится навчання класифікаторів на основі покрашенної версії початкового ввідного датасету. Уже тут можна побачити, що ефективність класифікації, хоча і не на багато, але підвищилася.

```
from sklearn.neural_network import MLPClassifier
MLP = MLPClassifier(hidden_layer_sizes=(1000, 300, 300), solver='adam', shuffle=False, tol = 0.0001)
```

```
MLP.fit(X_train_cordrop, Y_train)
mlp_finalscore = MLP.score(X_train_cordrop, Y_train)
```

```
first_model = {'Model': ['Linear Regression', 'Adaboost', 'RandomForest', 'ExtraTrees', 'GradientBoost'],
               'accuracy' : [lr_score, ab_score, rf_score, et_score, gb_score]}

result_df = pd.DataFrame(data = first_model)
result_df
```

	Model	accuracy
0	Linear Regression	0.334265
1	Adaboost	0.755000
2	RandomForest	0.754600
3	ExtraTrees	0.769600
4	GradientBoost	0.767100

Рисунок. 3.37 – Імпорт бібліотек для методу класифікації MPL. Тренування та розрахунок ефективності MPL та побудова таблиці з порівнянням ефективності основних методів класифікації.

Визначивши ефективність методів класифікації можна зробити висновки, що результати по навчанню за першим не модифікованим датасетом складає приблизно 75%, з вийнятком методу лінійної регрессії, що складає всього лише 34%.

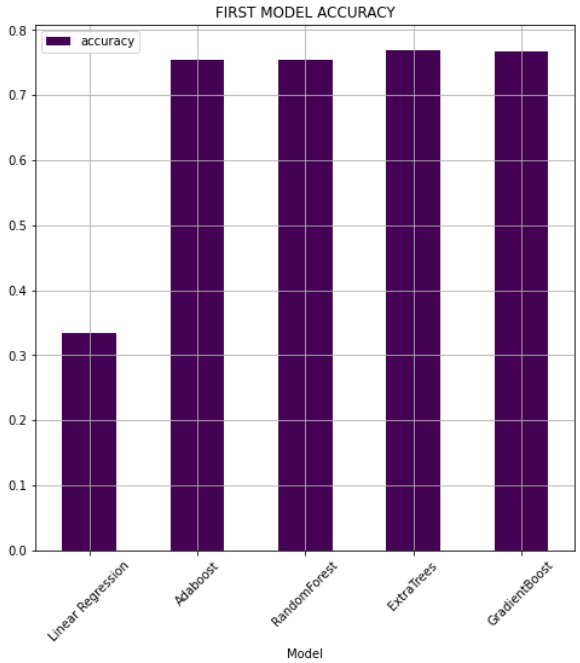

Рисунок. 3.38 – Графік порівняння ефективності методів класифікації за першим немодифікованним датасетом.

```
second_model = {'Model': ['Adaboost', 'RandomForest', 'ExtraTrees', 'GradientBoost'],
                'accuracy' : [ab2_score, rf2_score, et2_score, gb2_score]}

result_df = pd.DataFrame(data = second_model)
result_df
```

	Model	accuracy
0	Adaboost	0.7706
1	RandomForest	0.7717
2	ExtraTrees	0.7592
3	GradientBoost	0.7494

Рисунок 3.39 – Таблиця порівняння методів класифікації навченних за модифікованими датасетами X_test/learn_stdrop

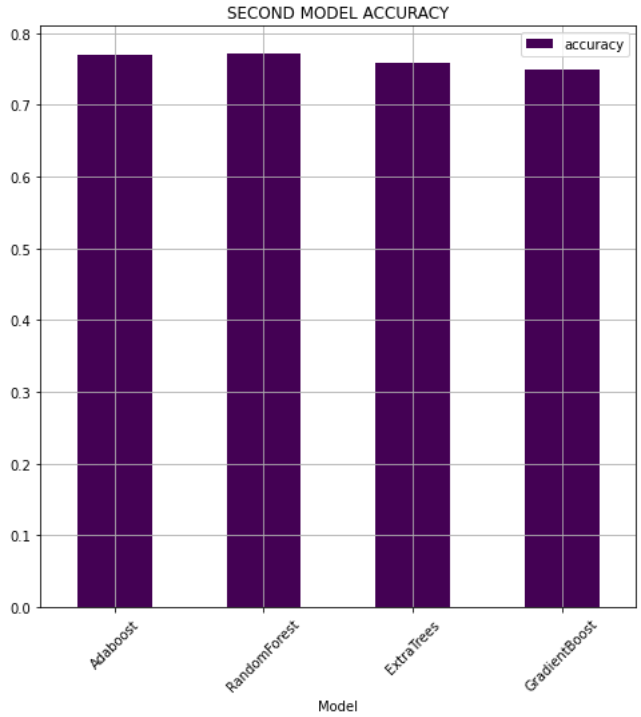

Рисунок 3.40 – Графік порівняння ефективності методів класифікації за другим датасетом навченим на модифікованому датасеті X_test/learn_stdrop

Проаналізувавши результати навчання методів класифікації по другій моделі можна побачити, що усі методи класифікації мають приблизно однакову ефективність, за виключенням алгоритмів AdaBoost та RandomForest. Їх ефективність складає 77% у порівнянні з навченими по першій моделі 75%.

```
final_model = {'Model': ['Linear Regression', 'Adaboost', 'RandomForest', 'ExtraTrees', 'GradientBoost', 'MLP'],
               'accuracy' : [lr_finalscore, ab_finalscore, rf_finalscore, et_finalscore, gb_finalscore, mlp_finalscore

result_df = pd.DataFrame(data = final_model)
result_df
```

	Model	accuracy
0	Linear Regression	0.339197
1	Adaboost	0.763600
2	RandomForest	0.752700
3	ExtraTrees	0.763300
4	GradientBoost	0.766000
5	MLP	0.905043

Рисунок 3.41 – Таблиця з порівнянням методів класифікації навчених на можифікованому датасеті X_train/test_cordrop

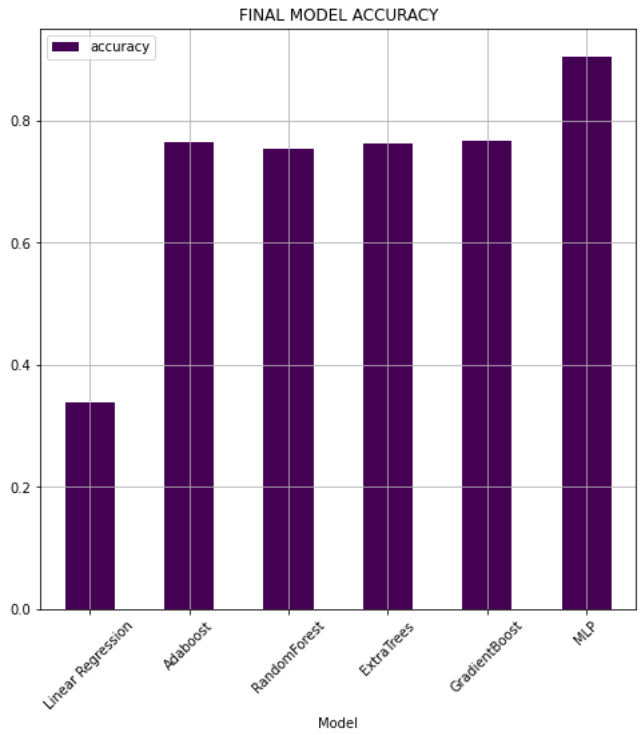

Рисунок 3.42 – Графік порівняння ефективності методів класифікації за фінальним датасетом навченим на модифікованому датасеті X_test/learn_cordrop

Навчання на фінальній моделі із алгоритмами AdaBoost ExtraTrees та GradientBoost показали результат приблизно 76%, що не значно, але краще, за результат першої базової моделі. Найвищий результат показав алгоритм MLP із результатом 90%.

Висновки за розділом 3

У ході написання 3го розділу було на практиці виконано із використанням інтерактивної Python консолі такі завдання:

1. Розроблено план модифікації моделей навчання, та створено блок-схему, що описує процесс тренування і модифікації. План модифікації включав в себе такі методи як заміна, вилучення малоінформативних стовбців, вибір та використання найбільш інформативних стовбців, визначено за допомогою градієнтного графіку та видалено високо кореляційні стовбці.

2. Розроблено механізм ідентифікації вразливостей використовуючи методи класифікації на основі нейронних мереж, навченних на модифікованих за планом датасетах.

3. Виконано порівняння методів класифікації на різних етапах модифікування вхідних данних. Найвищий результат показав алгоритм багатошарового персептрону (MPL) і має точність 90% у передбаченні потенційно шкідливого та загрозливого трафіку.

4. Використано методи для покращення результатів шляхом модифікування вхідних данних, побудовані вспоміжні методи для потенційного підвищення ефективності класифікаторів навченних за модифікованими вхідними данними.

ВИСНОВКИ

У дипломній роботі розв’язано актуальне завдання щодо створення механізму визначення мережевих загроз використовуючи класифікатори на основі нейронних мереж. В ході розв’язання поставлених задач були отримані наступні практичні результати:

1. Проаналізовано методи пасивного мережевого аналізу, механізми їх роботи, використання пасивного мережевого аналізу в системах безпеки, що складається з: SPI, MPI та DPI.

2. Для класифікації мережевих загроз були визначені методи класифікації мережевих загроз, що можуть використовувати перехоплений трафік – AdaBoost, RandomForest, ExtraTrees, GradientBoost та MLP.

3. В результаті порівняння ефективності методів класифікації мережевих загроз було визначено найкращій метод який можна використовувати для виконання поставленого завдання, що є MLP (з ефективністю 90,05%).

4. Покращено ефективність деяких методів класифікації шляхом модифікації данних, що надаються методам. А саме було виконано видалення стовбців з невеликою інформативністю, великою кореляцією та вибору виключно стовбців з найбільшим впливом.

5. Розроблено механізм визначення мережевих загроз, використовуючи методи класифікації заснованих на нейронних мережах та визначено найкращий метод за остаточними модифікованими данними, що є MPL.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Анализ сетевого трафика в режиме реального времени: обзор прикладных задач, подходов и решений. [Електронний ресурс]. – Режим доступу: https://www.ispras.ru/preprints/docs/prep_28_2015.pdf

2. Stephen Barish. Passive network analysis. [Електронний ресурс]. – Режим доступу: https://www.securitylab.ru/analytics/350448.php

3. Wireshark. [Електронний ресурс]. – Режим доступу: https://www.wireshark.org/

4. P0f. [Електронний ресурс]. – Режим доступу: https://lcamtuf.coredump.cx/p0f3/

5. Antonello, R. Deep packet inspection tools and techniques in commodity platforms: Challenges and trends / Rafael Antonello // Journal of Network and Computer Applications, 2012. – № 6. – С. 7–9. – Access: https://www.researchgate.net/publication/257489381_Deep_packet_inspection_tools_and_techniques_in_commodity_platforms_Challenges_and_trends

6. SPI [Електронний ресурс]. – Режим доступу: https://itlaw.wikia.org/wiki/Shallow_packet_inspection

7. Velea, R. Network Traffic Anomaly Detection Using Shallow Packet Inspection and Parallel K-means Data Clustering / Radu Velea // Studies in Informatics and Control, 2017. – № 4. – С. 12–19. – Access: https://www.researchgate.net/publication/321753578_Network_Traffic_Anomaly_Detection_Using_Shallow_Packet_Inspection_and_Parallel_K-means_Data_Clustering

8. David L. Cannon. CISA Certified Information Systems Auditor Study Guide, 2nd Edition / L. Cannon. David – N. Y.: ACM, 2008, 46 с.

9. I. Dubrawsky, “Firewall Evolution - Deep Packet Inspaction.” [Електронний ресурс]. – Режим доступу: http://www.securityfocus.com/infocus/1716, July 2003.

10. Fisk, M. An Analysis of Fast String Matching Applied to Content-based Forwarding and Intrusion Detection / Michael Fisk // Streaming Network Measurement &

Security, 2014. . – № 9. – С. 34–35. – Access: https://www.researchgate.net/publication/228366076_Applying_fast_string_matching_to_intrusion_detection

11. Schuff, D. Design alternatives for a high-performace self-securing ethernet network interface / Derek Schuff // Parallel and Distributed Processing Symposium, 2007. – № 19. – С. 51–52. – Access: https://www.researchgate.net/publication/224712583_Design_Alternatives_for_a_High-Performance_Self-Securing_Ethernet_Network_Interface

12. G. Gilder, Telecosm: How Infinite Bandwidth Will Revolutionize Our World / Glider, G. – M.: Kindle, 2000. – 336 с.

13. White paper on Deep Packet Inspection [Електронний ресурс]. – Режим доступу: https://www.ipoque.com/news-media/resources/whitepapers/deep-packet-inspection

14. IDS [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Intrusion_detection_system

15. Firewall Evolution - Deep Packet Inspection. [Електронний ресурс]. – Режим доступу: https://www.academia.edu/6937224/Firewall_evolution_deep_packet_inspection

16. Probabilistic alert correlation [Електронний ресурс]. – Режим доступу: https://cs.fit.edu/~pkc/id/related/valdes-raid01.pdf

17. Xu D. and Ning, P. Alert correlation through triggering events and common resources / Ding Xu and Peng Ning // “20th Annual Computer Security Applications Conference (6-10 Dec. 2004)” – Вип. 7 – Access: https://ieeexplore.ieee.org/document/1377243/

18. F. Valeur, G. Vigna, C. Kruegel, and R. A. Kemmerer, “A comprehensive approach to intrusion detection alert correlation” – Вип. 5 – Режим доступу: https://ieeexplore.ieee.org/document/1377243/authors#authors

19. Snort IDS [Електронний ресурс]. – Режим доступу: http://www.snort.org

20. Bleeding IDS [Електронний ресурс]. – Режим доступу: http://www.bleedingthreats.net

21. van Lunteren J. and Engbersen T. Fast and scalable packet classification / Jon van Lunteren and Tom Engbersen // IEEE Journal on Selected Areas in Communications, 2003. – № 4. – С. 11–16. – Access: https://ieeexplore.ieee.org/document/1197701

22. Gupta P. and McKeown, N. Algorithms for packet classification / Paul Gupta and McKeown Nick // “IEEE International Symposium on Industrial Electronics (5-8 July 2009)” – Вип. 3 – Access: https://ieeexplore.ieee.org/document/5215939

23. D. Taylor, Survey and taxonomy of packet classification techniques / Taylor David // ACM Computing Surveys, 2005. – № 37. – С. 238–275. – Access: https://dl.acm.org/doi/10.1145/1108956.1108958

24. PCRE -Perl Compatible Regular Expressions [Електронний ресурс]. – Режим доступу: “http://www.pcre.org/.”

25. Adaboost Algorythm [Електронний ресурс]. – Режим доступу: https://www.mygreatlearning.com/blog/adaboost-algorithm/

26. RandomForest Algorythm [Електронний ресурс]. – Режим доступу: https://builtin.com/data-science/random-forest-algorithm

27. ExtraTrees Algorythm [Електронний ресурс]. – Режим доступу: https://medium.com/@namanbhandari/extratreesclassifier-8e7fc0502c7

28. GradientBoosting Algorythm [Електронний ресурс]. – Режим доступу: https://stackabuse.com/gradient-boosting-classifiers-in-python-with-scikit-learn/

29. MLP Algorythm [Електронний ресурс]. – Режим доступу: https://wiki.pathmind.com/multilayer-perceptron

30. Функція .head() [Електронний ресурс]. – Режим доступу: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.head.html

31. Функція .value_counts() [Електронний ресурс]. – Режим доступу: https://pandas.pydata.org/pandasdocs/stable/reference/api/pandas.Series.value_counts.html

32. Бібліотека sklearn [Електронний ресурс]. – Режим доступу: https://scikit-learn.org/stable/user_guide.html

33. Numpy [Електронний ресурс]. – Режим доступу: https://numpy.org/doc/stable/reference/generated/numpy.std.html

34. Функція .nsmallest() [Електронний ресурс]. – Режим доступу: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.nsmallest.html

35. Research on Packet Inspection Techniques [Електронний ресурс]. – Режим доступу: http://www.ijstr.org/final-print/nov2019/Research-On-Packet-Inspection-Techniques.pdf

36. АНАЛИЗ IP-ТРАФИКА МЕТОДАМИ DATA MINING. ПРОБЛЕМА КЛАССИФИКАЦИИ [Електронний ресурс]. – Режим доступу: https://icmmg.nsc.ru/sites/default/files/pubs/sh2012-4.pdf

37. Машинное обучение для анализа и классификации шифрованного сетевого трафика [Електронний ресурс]. – Режим доступу: https://scm.etu.ru/assets/files/2020/scm20/papers/4/238.pdf

38. Анализ локальной сети пассивным методом [Електронний ресурс]. – Режим доступу: https://nestor.minsk.by/kg/2009/03/kg90301.html

39. Метод идентификации киберпреступников, использующих инструменты сетевого анализа информационных систем с применением технологий анонимизации. [Електронний ресурс]. – Режим доступу: https://cyberleninka.ru/article/n/metod-identifikatsii-kiberprestupnikov-ispolzuyuschih-instrumenty-setevogo-analiza-informatsionnyh-sistem-s-primeneniem-tehnologiy

40. Исследование угроз безопасности и атак в сетях SS7 [Електронний ресурс]. – Режим доступу: https://www.ptsecurity.com/ru-ru/research/analytics/ss7-vulnerability-2018/

ДОДАТОК А

Лістинг програмного коду використанного у третьому розділі дипломної роботи:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib  as matplot
import numpy as np

import re
import sklearn

import warnings
warnings.filterwarnings("ignore")

%matplotlib inline

df_train = pd.read_csv('./input/Train_data.csv')
df_test = pd.read_csv('./input/test_data.csv')
df_test = df_test.drop('Unnamed: 0',axis=1)
df_train.head()
df_test.head()

df_test["xAttack"].value_counts()
X_train = df_train.drop('xAttack', axis=1)
Y_train = df_train.loc[:,['xAttack']]
X_test = df_test.drop('xAttack', axis=1)
```

```
Y_test = df_test.loc[:,['xAttack']]
X_train.head(10)
Y_train.head(10)
from sklearn import preprocessing
from sklearn.preprocessing import OneHotEncoder
le = preprocessing.LabelEncoder()
enc = OneHotEncoder()
lb = preprocessing.LabelBinarizer()
X_train["protocol_type"].value_counts()
X_train['protocol_type'] = le.fit_transform(X_train['protocol_type'])
X_test['protocol_type'] = le.fit_transform(X_test['protocol_type'])
X_train.head(10)
X_train["protocol_type"].value_counts()
Y_train['xAttack'].value_counts()
Y_train['xAttack'] = le.fit_transform(Y_train['xAttack'])
lb.fit_transform(Y_train['xAttack'])

Y_test['xAttack'] = le.fit_transform(Y_test['xAttack'])
lb.fit_transform(Y_test['xAttack'])

Y_train['xAttack'].value_counts()
Y_train.describe()
con_list = ['protocol_type', 'service', 'flag', 'land', 'logged_in', 'su_attempted',
'is_host_login', 'is_guest_login']
con_train = X_train.drop(con_list, axis=1)


stdtrain = con_train.std(axis=0)
std_X_train = stdtrain.to_frame()
std_X_train.nsmallest(10, columns=0).head(10)
```

```
X_train = X_train.drop(['num_outbound_cmds'], axis=1)
X_test = X_test.drop(['num_outbound_cmds'], axis=1)

df_train = pd.concat([X_train, Y_train], axis=1)
stdrop_list = ['urgent', 'num_shells', 'root_shell',
    'num_failed_logins', 'num_access_files', 'dst_host_srv_diff_host_rate',
    'diff_srv_rate', 'dst_host_diff_srv_rate', 'wrong_fragment']

X_test_stdrop = X_test.drop(stdrop_list, axis=1)

X_train_stdrop = X_train.drop(stdrop_list, axis=1)

df_train_stdrop = pd.concat([X_train_stdrop, Y_train], axis=1)

df_train_stdrop.head()
from sklearn import linear_model
LR = linear_model.LinearRegression()
LR.fit(X_train,  Y_train)
from sklearn.ensemble  import AdaBoostClassifier
from sklearn.ensemble  import RandomForestClassifier
from sklearn.ensemble  import BaggingClassifier
from sklearn.ensemble  import ExtraTreesClassifier
from sklearn.ensemble  import GradientBoostingClassifier

from sklearn.tree import DecisionTreeClassifier

AB = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(),
n_estimators=100, learning_rate=1.0)
RF = RandomForestClassifier(n_estimators=10, criterion='entropy',
max_features='auto', bootstrap=True)
```

```
ET = ExtraTreesClassifier(n_estimators=10, criterion='gini', max_features='auto', bootstrap=False)
GB = GradientBoostingClassifier(loss='deviance', learning_rate=0.1, n_estimators=200, max_features='auto')
AB.fit(X_train, Y_train)
RF.fit(X_train, Y_train)
RF_feature = RF.feature_importances_

rf_score = RF.score(X_test, Y_test)

print('RandomForestClassifier Score: %.3f %%' % rf_score)
ET.fit(X_train, Y_train)
ET_feature = ET.feature_importances_
GB.fit(X_train, Y_train)
cols = X_train.columns.values

feature_df = pd.DataFrame({'features': cols,
                           'AdaBoost' : AB_feature,
                           'RandomForest' : RF_feature,
                           'ExtraTree' : ET_feature,
                           'GradientBoost' : GB_feature
                          })
feature_df.head(8)
from matplotlib.ticker import MaxNLocator
from collections import namedtuple

graph = feature_df.plot.bar(figsize = (18, 10), title = 'Feature distribution', grid=True, legend=True, fontsize = 15,
                            xticks=feature_df.index)
graph.set_xticklabels(feature_df.features, rotation = 80)
```

```
a_f = feature_df.nlargest(12, 'AdaBoost')
e_f = feature_df.nlargest(12, 'ExtraTree')
g_f = feature_df.nlargest(12, 'GradientBoost')
r_f = feature_df.nlargest(12, 'RandomForest')
result = pd.concat([a_f, e_f, g_f, r_f])
result = result.d
rop_duplicates()
result
selected_features = result['features'].values.tolist()
selected_features
AB.fit(X_train_stdrop, Y_train)
ab2_score = AB.score(X_test_stdrop, Y_test)
print('AdaBoostClasifier Score: %.3f %%' % ab2_score)
RF.fit(X_train_stdrop, Y_train)
ET.fit(X_train_stdrop, Y_train)
GB.fit(X_train_stdrop, Y_train)
X_train_ens = X_train[selected_features]
X_train_ens.head()
X_test_ens = X_test[selected_features]
X_test_ens.head()
sample = X_train_ens[:10000]

colormap = plt.cm.viridis
plt.figure(figsize=(20,  20))
sns.heatmap(sample.astype(float).corr(),  linewidths=0.1,  vmax=1.0,  square=True,
cmap=colormap, annot=True)
selected2 = ['dst_host_serror_rate', 'serror_rate']
X_train_cordrop = X_train_ens.drop(selected2, axis=1)
X_train_cordrop.describe()
X_test_cordrop = X_test_ens.drop(selected2, axis=1)
```

```
X_test_cordrop.describe()
AB.fit(X_train_cordrop, Y_train)
ab_finalscore = AB.score(X_test_cordrop, Y_test)
print('AdaBoostClassifier_final Score: %.3f %%' % ab_finalscore)
RF.fit(X_train_cordrop, Y_train)
rf_finalscore = RF.score(X_test_cordrop, Y_test)
print('RandomForestClassifier_final Score: %.3f %%' % rf_finalscore)
ET.fit(X_train_cordrop, Y_train)
et_finalscore = ET.score(X_test_cordrop, Y_test)
print('ExtraTreesClassifier_final Score: %.3f %%' % et_finalscore)
GB.fit(X_train_cordrop, Y_train)
gb_finalscore = GB.score(X_test_cordrop, Y_test)
print('GradientBoostClassifier_final Score: %.3f %%' % gb_finalscore)
LR.fit(X_train_cordrop, Y_train)
lr_finalscore = LR.score(X_test_cordrop, Y_test)
print('LinearRegression_final Score: %.3f %%' % lr_finalscore)
from sklearn.neural_network import MLPClassifier
MLP = MLPClassifier(hidden_layer_sizes=(1000, 300, 300), solver='adam',
shuffle=False, tol = 0.0001)
MLP.fit(X_train_cordrop, Y_train)
mlp_finalscore = MLP.score(X_train_cordrop, Y_train)
first_model = {'Model': ['Linear Regression', 'Adaboost', 'RandomForest',
'ExtraTrees', 'GradientBoost'],
        'accuracy' : [lr_score, ab_score, rf_score, et_score, gb_score]}

result_df = pd.DataFrame(data = first_model)
result_df
r1 = result_df.plot(x='Model', y='accuracy', kind='bar', figsize=(8, 8), grid=True,
title='FIRST MODEL ACCURACY', colormap=plt.cm.viridis,
        sort_columns=True)
```

```
r1.set_xticklabels(result_df.Model, rotation = 45)
second_model = {'Model': ['Adaboost', 'RandomForest', 'ExtraTrees', 'GradientBoost'],
                'accuracy' : [ab2_score, rf2_score, et2_score, gb2_score]}

result_df = pd.DataFrame(data = second_model)
result_df
r2 = result_df.plot(x='Model', y='accuracy', kind='bar', figsize=(8, 8), grid=True, title='SECOND MODEL ACCURACY', colormap=plt.cm.viridis,
                sort_columns=True)
r2.set_xticklabels(result_df.Model, rotation = 45)
final_model = {'Model': ['Linear Regression', 'Adaboost', 'RandomForest', 'ExtraTrees', 'GradientBoost', 'MLP'],
                'accuracy' : [lr_finalscore, ab_finalscore, rf_finalscore, et_finalscore, gb_finalscore, mlp_finalscore]}

result_df = pd.DataFrame(data = final_model)
result_df
r3 = result_df.plot(x='Model', y='accuracy', kind='bar', figsize=(8, 8), grid=True, title='FINAL MODEL ACCURACY', colormap=plt.cm.viridis,
                sort_columns=True)
r3.set_xticklabels(result_df.Model, rotation = 45)
```

ДОДАТОК Б

Апробація дипломної роботи на на XXI Всеукраїнській науково-технічній конференції молодих вчених, аспірантів та студентів у Університеті Інформатики і прикладних знань м. Одесса.

ПАСИВНИЙ МЕРЕЖЕВИЙ АНАЛІЗ ТА ЗАСОБИ ЙОГО ВДОСКОНАЛЕННЯ. ЖОЛНЕР І.Д., МИРУТЕНКО Л.В., ШЕСТАК Я.В. Розділ 2, ст. 63

ПАСИВНИЙ _МЕРЕЖЕВИЙ _АНАЛІЗ _ТА _ЗАСОБИ _ЙОГО _ВДОСКОНАЛЕННЯ _

ЖОЛНЕР І. Д., МИРУТЕНКО Л. В., ШЕСТАК Я.В.
(*ivzholner@gmail.com, myrutenko.lara@gmail.com, lucenko.y@ukr.net*),
Київський національний університет ім. Тараса Шевченка

Із розвитком мережевих технологій та збільшенням обсягів інформації, що передається через мережу, виникає потреба в поглибленому аналізі трафіку. Було розглянуто пасивний метод аналізу мережі, проаналізовано його переваги та недоліки в порівнянні з активним методом. Для усунення існуючих недоліків даного методу було запропоновано використання нейронних мереж, які, вірно відкалібрувавши вагу нейронів, зможуть робити точні передбачення на рахунок рівня загрози подій у мережі, та використання ідентифікації (fingerprinting) по заголовкам пакетів отримючи інформацію про ОС користувача, дивлячись на те яким чином конфігурований зміст відправленого пакету.

У сучасному світі важко уявити своє життя без інформаційних технологій, включаючи мережеві технології, які, в свою чергу, потребують постійного розвитку та вдосконалення. Це призводить до ускладнення їх структури та збільшення розмірів, а також до потреби розробки інструментів, що дозволять як локалізувати проблеми, які виникають у мережах, так і проаналізувати причини їх появи. Таку задачу на сьогоднішній день вирішують мережеві аналізатори, перевагами використання яких є можливість проводити накопичення, обробку, класифікацію, контроль і модифікацію мережевих пакетів в залежності від їх вмісту в реальному часі. Але при використанні наряду з перевагами аналізатори мають і ряд недоліків. Однією з основних проблем є те, що для використання таких аналізаторів треба «влучно» та актуально вибрати місце для розміщення мережевих сенсорів, які будуть зчитувати необхідну інформацію та здійснювати при цьому мінімальний вплив на пропускну здатність критичних точок системи. Крім того, слід звернути увагу, що навіть при вдалому виборі розміщення, спеціалістам з кібербезпеки (security analyst) необхідно також проаналізувати отриману інформацію та визначити рівень загрози кожної неправомірної дії.
Таким чином, можна зробити висновок, що навіть маючи персонал, який буде займатись розслідуванням та нейтралізацією загроз, можлива неправильна ідентифікація дій користувачів у мережі як загроз, або ж навпаки (false positive, false negative). Даний недолік можна вирішити за допомогою технологій побудови нейронних мереж на основі дата-сету для подальшого використання нейронної мережі в ідентифікації рівня загроз різних мережевих подій (events), які, в залежності від степені загрози, зможуть запропонувати можливі вектори нападу та захисту. Також хотілось би відмітити, що у сканерах безпеки майже не реалізуються методи ідентифікації користувачів по різним заголовкам пакетів (fingerprinting). Дана технологія могла б підвищити ефективність розслідувань подій, а також допомогти в ідентифікації правопорушників (або хоча б пристроїв, які вони

використовують). Запровадження таких вдосконалень можливо на рівні середовища накопичення обробки та класифікації перехоплених пакетів.

Слід звернути увагу на те, що методи пасивного мережевого аналізу працюють так, як і активні методи побудови карт. Пасивні методи ґрунтуються на сценарії «запит-відповідь», вони покладаються на чужій запит, а потім збирають відповіді. В активному сценарії об'єкт відповідає на запит додатка, який будує карти мережі, що є ефективним. У сценарії пасивного мережевого аналізу об'єкт відповідає на запити в результаті нормального функціонування мережевих додатків. В обох випадках отримуються дані про вибір потрібного порту і службах, потоках з'єднань, інформацію про час, за якими можна зробити припущення про робочі характеристики мережі. Пасивний метод дозволяє також зробити те, що неможливо зробити за допомогою активного методу: можна бачити мережу з точки зору користувача і вивчати поведінку додатків в ході звичайних операцій. Такі аналізатори зазвичай перехоплюють мережевий трафік, сприймаючи не адресовані їм пакети даних, переводячи свою мережеву картку в нерозбірливий режим. Або ж використовують встановлені у робочій мережі датчики, для перехоплення трафіку та подальшого їх логування.

Але пасивний аналіз також має свої недоліки. Для проведення пасивного аналізу потрібно вставити апаратний або програмний датчик в досліджувану мережу. Датчики повинні бути розміщені в топології мережі так, щоб через них проходив корисний трафік, що не є тривіальним завданням в сучасних мережах. І нарешті, інструментарій для пасивного аналізу набагато менш розвинений, ніж традиційні методи активного аналізу, тому що вимагають значних зусиль від аналітика для розміщення датчиків, збору даних і аналізу результатів. Такі фактори формують виникнення наведених недоліків.

Виправлення відбувається при використанні глибоких нейронних мереж, або ж нейронних мереж натренованих на трафіку, що циркулює у мережі підприємства. Використовуючи цей метод, аналіз та ідентифікація загроз може бути значно спрощена за допомогою нейронної мережі. Такі мережі можуть бути натреновані за допомогою використання різних методів. Наприклад: метод тренування по дельта-правилу, MNIST або по базі данних (Modified National Institute of Standards and Technology), з «вчителем» або без нього. Одним з таких методів є написання штучного інтелекту, який, навчений на реальному мережевому трафіку, буде генерувати різноманітний трафік, включаючи шкідливий а також корегувати вагу нейронів. А потім використання цього штучного інтелекту для генерування дата-сету тренування нейронної мережі, що буде спрощувати аналіз. Другий метод – написання такого дата-сету вручну, враховуючи потреби та найчастіші загрози безпеці підприємства, використовуючи базу даних загроз, яка містить вже відомі загрози всередині мережевого трафіку, що аналізуємо.

Таким чином, ми розглянули метод аналізу за допомогою переведення мережевої карти в нерозбірливий режим та метод розставляння сенсорів при пасивному мережевому аналізу. Виявлені деякі недоліки пасивного аналізу та запропоновано методи їх вирішення, а саме для вирішення проблеми аналізу загроз безпеці надаються методи використання нейронної мережі, як допоміжний інструмент у аналізі. Запропоновано використання ідентифікації користувачів мережі по заголовкам пакетів, що вони відправляють, для полегшення розслідувань та ідентифікації пристроїв з яких відбувалися неправомірні дії.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

[1] Alexander J. Beecroft, Passive Fingerprinting of Computer Network Reconnaissance Tools, Monterey, CA 93943-5000, 2009, 89 p.

[2] A. Bremler-Barr, Y. Harchol, D. Hay, Y. Koral, Deep packet inspection as a service, 2014, p. 271.

[3]: Пассивный анализ сети. [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://www.securitylab.ru/analytics/350448.php

[4] Исследование угроз безопасности и атак в сетях [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https:// .ptsecurity.com/ru-ru/research/analytics/ss7-vulnerability-2018/

[5] Colin J. Bennett, Andrew Clement, Kate Milberry, Introduction to Cyber-Surveillance. Cyber Surveillance in Everyday Life, 2012, 21 p.

[6] Net ork Intrusion Detection Signatures. [Електронний ресурс]. – 2015. – Режим доступу до ресурсу: http:// .symantec.com/connect/articles/net ork-intrusion-detection-signatures-partfive.